1992

Decomposition and Parallel Solution of Network-Structured Optimization Problems

Mustafa Çelebi Pinar

Decomposition and Parallel Solution of Network-Structured Optimization Problems

Abstract

Optimization problems characterized by an *embedded* network structure in the constraint matrix are frequently used as a modeling tool in many diverse application areas such as transportation, logistics, finance, telecommunications and so on. We develop techniques that exploit the special structure present in this class of problems. A solution methodology where we propose to place the complicating or *side* constraints into the objective function using the 1-norm penalty function is developed. Thus we obtain a nondifferentiable penalty problem with network constraints. We develop smoothing techniques for the 1-norm penalty function and establish their properties. By using a quadratic smoothing term we obtain a nonlinear nonseparable problem with network constraints. The penalty problem is solved iteratively using a decomposition technique based on a simplicial decomposition of the network constraint set. This decomposition scheme induces separability in the objective function through linearization in the subproblem phase and a nonlinear nonseparable master problem is solved based on the information obtained from the subproblem phase. We develop two specializations of the algorithm: (1) for the network flow problem with side constraints, (2) for the multicommodity flow problem. We present numerical results with Patient Distribution System (PDS) multicommodity flow problems and with network flow problems with side constraints derived from matrix estimation problems and the NETLIB Linear Programming Library problems.

The decomposition is particularly suitable for vector multiprocessor systems. We develop a parallel implementation of the linear-quadratic penalty algorithm. Numerical results with a set of large linear multicommodity network flow problems drawn from a military planning application are presented. The impact of parallel decomposition is investigated using a CRAY Y-MP supercomputer system and a Connection Machine CM-2. The parallelism is exploited both at the *tightly coupled* linear algebra level in the master problem and at a *loosely coupled* level with the network subproblems. Data-level parallel computing is explored on a massively parallel SIMD system, the Connection Machine CM-2.

As alternatives to simplicial decomposition, two decomposition techniques (1) based on the truncated Newton algorithm (2) on a cyclic decomposition are also considered. The truncated Newton based decomposition is developed for the single commodity case.

We conclude the thesis with an application from Naval Personnel Assignment formulated as a large network model with side constraints and solved using the penalty algorithm.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

First Advisor

Stavros A. Zenios

Decomposition and Parallel Solution of Network-Structured Optimization Problems

MUSTAFA ÇELEBI PINAR

A DISSERTATION

IN

SYSTEMS

Presented to the Faculties of the University of Pennsylvania in
Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

1992

Stavros A. Zenios

Supervisor of Dissertation

Edward K. Morlok

Graduate Group Chairperson

Acknowledgements

I want to express my gratitude to a number of individuals who provided help and support during my doctoral studies. Special thanks are due to:

Stavros A. Zenios, the supervisor of this dissertation. Stavros provided support for the project for four years. He supplied many ideas and encouragement at moments of distress. Through his energy and relentless mentoring he pushed me to the finish line. For everything he has done for me, I remain grateful.

Monique Guignard-Spielberg. Monique taught me many things on optimization in the classroom and outside on numerous occasions. She was always willing to listen and help. For everything she has done, I am grateful. Mes très sincères remerciements!

Edward K. Morlok and **Irvin Lustig.** They took time from their schedules to serve as readers of this dissertation. For their constructive criticism and valuable feedback, I am grateful.

Ron Dembo. Ron served as the spiritual father of the smoothing ideas developed in this thesis. He made his expertise on nonlinear programming available at the beginning of the project. For his help, I remain grateful.

Patrick Harker and **Keith Ross.** They served as readers in the proposal phase of this dissertation. Keith was a true friend from my first day in the Engineering School. Pat provided several suggestions during the proposal phase.

I met some of my best friends during those four years. I want to mention Soren, Vipul, Leonora, Manos, Kooka, Rui-Jin, Nuri and Zeynep. And, also everyone in Decision Sciences especially Marge, Cynthia, Kim and Kass. I thank them all for their friendship and courtesy.

Finally, I dedicate this thesis to the dearest people in my life, my mother Suzan, my late father Ertan, late Erhan Dilligil, and Leyla. They are my raison d'être.

Abstract

Optimization problems characterized by an *embedded* network structure in the constraint matrix are frequently used as a modeling tool in many diverse application areas such as transportation, logistics, finance, telecommunications and so on. We develop techniques that exploit the special structure present in this class of problems. A solution methodology where we propose to place the complicating or *side* constraints into the objective function using the 1-norm penalty function is developed. Thus we obtain a nondifferentiable penalty problem with network constraints. We develop smoothing techniques for the 1-norm penalty function and establish their properties. By using a quadratic smoothing term we obtain a nonlinear nonseparable problem with network constraints. The penalty problem is solved iteratively using a decomposition technique based on a simplicial decomposition of the network constraint set. This decomposition scheme induces separability in the objective function through linearization in the subproblem phase and a nonlinear nonseparable master problem is solved based on the information obtained from the subproblem phase. We develop two specializations of the algorithm: (1) for the network flow problem with side constraints, (2) for the multicommodity flow problem. We present numerical results with Patient Distribution System (PDS) multicommodity flow problems and with network flow problems with side constraints derived from matrix estimation problems and the NETLIB Linear Programming Library problems.

The decomposition is particularly suitable for vector multiprocessor systems. We develop a parallel implementation of the linear-quadratic penalty algorithm. Numerical results with a set of large linear multicommodity network flow problems drawn from a military planning application are presented. The impact of parallel decomposition is investigated using a CRAY Y–MP supercomputer system and a Connection Machine CM-2. The parallelism is exploited both at the *tightly coupled* linear algebra level in the master problem and at a *loosely coupled* level with the network subproblems. *Data-level* parallel computing is explored on a massively parallel SIMD system, the Connection Machine CM-2.

As alternatives to simplicial decomposition, two decomposition techniques (1) based on the truncated Newton algorithm (2) on a cyclic decomposition are also considered. The truncated Newton based decomposition is developed for the single commodity case.

We conclude the thesis with an application from Naval Personnel Assignment formulated as a large network model with side constraints and solved using the penalty algorithm.

Contents

List of Figures

Chapter 1

Introduction and Background

The objective of this research is to design, analyze, and implement decomposition algorithms for large scale optimization problems with embedded network structures. The central theme of the material presented here is the design of a decomposition method based on the notion of linear-quadratic smooth penalty (LQP) functions. Traditionally, the constrained network flow problems have been attacked using basis partitioning or Lagrangean/subgradient methods. These methods proved to be effective for problems where only a few non-network constraints are present. In this thesis, we follow a different route. We develop a decomposition method based on the use of a smooth penalty function to eliminate the side (non-network) constraints from the constraint set and thus obtain a problem with flow conservation constraints. A sequence of penalty problems is solved to get a solution to the original problem. The efficiency of the penalty method hinges upon efficient solution of the nonlinear network penalty problems.

The first part of the thesis is devoted to the development of a smooth penalty algorithm for network-structured problems. To eliminate non-network constraints, we consider the 1-norm exact penalty function, Luenberger [1970]. However, this function is not differentiable everywhere. To obtain a differentiable penalty problem we consider a smooth approximation to the 1-norm exact penalty function. Properties of the approximation are analyzed and an approximate exactness property is established. Bounds on the approximation error are developed. Having a differentiable penalty function at hand, we seek to design efficient algorithms for the solution of the resulting network penalty problem. We develop two specializations of the penalty algorithm, (1) for multicommodity flow problems (2) for network

problems with side constraints.

The goal of this thesis is to develop effective and efficient solution methodologies for a broad class of large-scale optimization problems. Although the development here is confined to the domain of network structured problems, the methodology developed remains applicable to a larger class of problems where there is exploitable structure. At a more technical level, this thesis seeks answers to the following questions:

1 Are exact penalty functions useful computational tools for large scale optimization?

2 For what classes of problems the penalty function based decomposition developed in this study perform best?

3 Can parallel processing have a substantial impact on the solution of special structured problems?

With respect to the above questions, this thesis touches three seemingly distinct subjects: penalty functions, network optimization and parallel computing. It is rather surprising to find very few articles on applications of exterior penalty functions to large scale optimization problems. On the network optimization side, the constrained network problems have been solved using basis partitioning or Lagrangean techniques and Dantzig-Wolfe decomposition in the multicommodity case. The application of penalty functions to the solution of constrained network flow problem appears to be non-existent except for a few recent articles, see Brown et al. [1989]. The contents of this thesis bring together techniques from penalty function literature and network optimization to take advantage of the rich sparsity structure that the network problems possess. In this respect, it should be of interest to both communities. It also considers alternative forms of parallel computations for efficient solution of large instances.

Due to the quadratic term in the smooth penalty function, the penalty objective function becomes non-separable. In the case of the linear multicommodity network flow problem, the original linear problem is replaced by a non-trivial nonlinear problem. However, the block angular structure of the constraint matrix motivates the choice for a method that would induce separability in the objective function. There are various directions that can be pursued to achieve this goal. One alternative for the solution of the penalty problem is to use a linearization technique. This scheme induces separability in the objective function

and the problem decomposes into independent network flow problems for each commodity. Linearization based decomposition can be achieved using the Frank-Wolfe algorithm or variants thereof, such as simplicial decomposition, von Hohenbalken [1977]. This decomposition scheme fits naturally into *coarse grain* parallel architectures as well as *fine-grain* architectures. This is the approach taken in this research for the solution of the penalty problem. Numerical results reported in this thesis using test problems from a Military Airlift Command application indicate the effectiveness of this approach. We also specialize the linear-quadratic penalty technique to solve constrained matrix estimation problems and NETLIB linear programming problems. The algorithm remains robust and efficient for most of the problems solved from this class.

Another alternative for the solution of network penalty problem is to use a reduced gradient type method or a truncated Newton method. The truncated Newton method can take advantage of the special structure of the constraint set by identifying independent descent cycles and operating simultaneously on these cycles. A decomposition technique based on this idea has been developed and tested for the single commodity case. A description of the techniques and numerical results on a vector multiprocessor Alliant FX/8 are given in this thesis.

In the second part of the thesis we investigate parallel decomposition opportunities using the LQP algorithm. By virtue of linearization, the subproblem phase in simplicial decomposition consists of solving as many linear network flow problems as there are commodities. The master problem phase involves dense linear algebra operations which are suitable for parallel computations. Based on these observations a parallel implementation on a CRAY Y-MP shared memory multiprocessor system is developed. The linear network flow problems are solved in parallel using a network simplex algorithm. Very encouraging numerical results are given on problems with sizes of up to 150,000 variables.

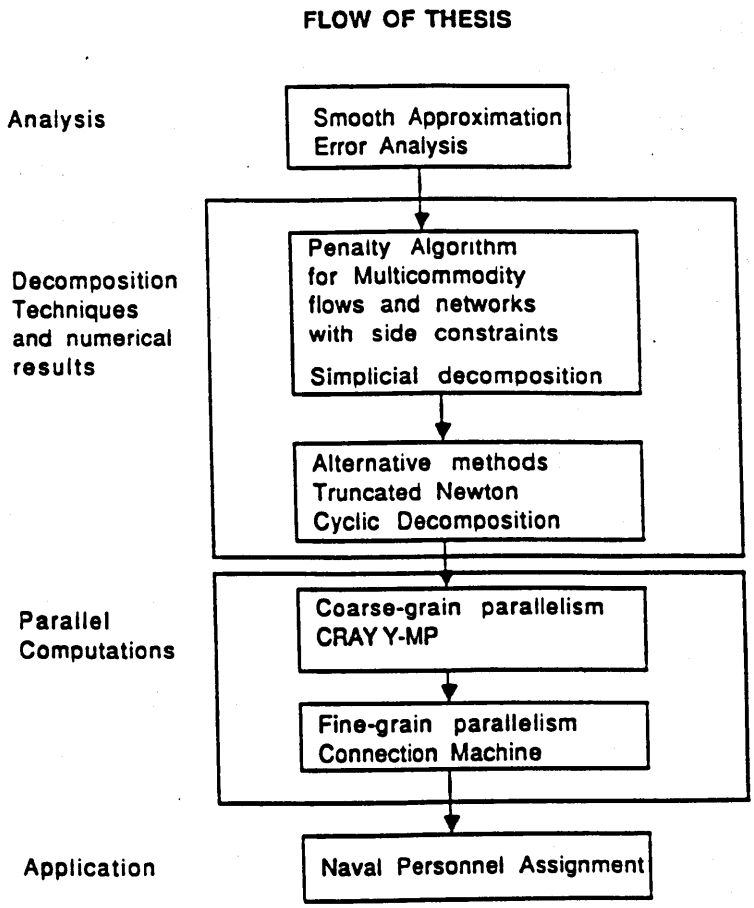
An interesting question is whether the linear-quadratic penalty technique can be implemented on a massively parallel Connection Machine system. We give an affirmative answer to this question in this thesis. The Connection Machine system is based on a SIMD (Single Instruction Multiple Data) or *data-level* parallel computing paradigm where a large number of processors execute the same operation on large ammounts of data in a synchronized manner. To what extent this parallel computing scheme is applicable to the solution of multicommodity flow problems using the linear quadratic penalty function is unclear.

We address this research question in this thesis. We investigate the potential of a leading data-level parallel computer system, the Connection Machine CM-2, for the solution of multicommodity flow problems in contrast to our work on a *control-level* parallel supercomputer, the CRAY Y-MP where fewer parallel processors can operate independently on different portions of data in an asynchronous fashion. In this respect We remark that not only the dense linear algebra operations of the master problem phase can be performed on the Connection Machine but also the subproblem phase can be performed on the Connection Machine using a combination of proximal point and row-action algorithms, Nielsen and Zenios [1991]. Since row-action algorithms require strict convexity, a vanishing proximal term is added to the objective function to solve the linear network problems on the Connection Machine. Preliminary computational results are given.

In the final part of the thesis we develop and solve an application drawn from a Navy manpower planning problem using the linear-quadratic penalty technique. The problem consists of maximizing Navy fleet readiness subject to personnel availability constraints. The problem is posed as a constrained network flow problem. The model also involves a side variable. It attains very large sizes and defies solution with existing network codes. Very large instances of this problem are solved and results are reported in Chapter 7 along with the model and a specialization of the linear-quadratic penalty technique to its solution.

Finally, a word on what is to follow and organization. In section 1.1, we give an overview of literature on multicommodity flows. For penalty functions and network problems with side constraints, brief reviews are given at the beginning of the chapters or in the appropriate section. We study the smooth approximation to the 1-norm exact penalty function in Chapter 2. We introduce and describe the linear-quadratic penalty method in Chapter 3 and specialize the algorithm for (1) multicommodity flows, and (2) for network flows with side constraints. We present computational results with a set of large linear multicommodity network flow problems drawn from a military application and constrained matrix estimation problems as well as NETLIB problems. In Chapter 4 we study alternative parallel decomposition techniques for the solution of the network penalty problem. The first technique we consider is a truncated Newton algorithm specialized for single commodity nonlinear network problems. Computational results with a set of nonlinear network problems are given. We then consider an alternative technique to solve the network penalty problem that we term *cyclic decomposition* and investigate its convergence properties. Coarse grain parallel

decompositions with the linear quadratic penalty method are discussed in Chapter 5 and computational results on a CRAY Y–MP vector multiprocessor are given. In Chapter 6, we discuss data-level parallelism for the solution of multicommodity flow problems and details of an implementation on a Connection Machine CM–2 system are presented. We present an application of the linear-quadratic penalty technique to a Naval Personnel Assignment problem in Chapter 7. The report concludes in Chapter 8 with extensions and future research issues. A flowchart of the thesis organization is given below.



1.1 A Review of Algorithms for Multicommodity Network Flows

In this section we will briefly review algorithms previously developed for the solution of multicommodity network flow problems. Multicommodity network flows are used as a modeling tool in many applications that arise in such diverse areas as financial modeling, telecommunications, logistics and transportation among many others. The problem usually consists of finding minimum cost flows of multiple commodities over a network with given demand and supply requirements and link capacity restrictions. The presence of link capacities translate into a set of constraints known as *Generalized Upper Bound* (GUB) constraints which complicate the structure of a problem which would otherwise have a block angular structure. A different version of the multicommodity flow problem is where a nonlinear congestion function is used as the objective function. In this model, the capacity constraints are absent but the coupling terms in the objective function still makes the problem non-separable, see for instance Bertsekas and Gafni [1983].

The above features make the considerable amount of tools and knowledge available from network optimization literature hardly applicable to the multicommodity network flow problem. Furthermore, the problem instances that are encountered as real applications tend to be fairly large, which prohibits the use of well–known algorithms like the simplex method. However, with the emergence of interior point methods, these problems become tractable, see for instance the paper by Lustig et al. [1990], although at the expense of high computation times. It is, therefore, a worthwhile effort to consider methods that would take advantage of the embedded block angular structure of these problems. Shared memory multiprocessor architectures constitute an ideal computing environment for the implementation of algorithms which would allow the decomposition of the multicommodity network flow problem into min-cost network flow problems of equal size for each commodity. This would also allow the use of network optimization tools which have been developed over the past twenty years, see for example the text by Kennington and Helgason [1980].

We discuss briefly classical methods based on basis partitioning and decomposition. An extensive treatment of these methods are available in numerous sources, see for example Kennington [1978], and will not be given here. We intend rather to focus on recently developed methods for the solution of large multicommodity network flow problems. In

particular we will describe a decomposition method based on logarithmic barrier functions by Schultz and Meyer [1990], an algorithm based on the notion of coercion functions by Zenios, Qi and Armstrong [1991] and review approaches based on augmented Lagrangian penalty functions.

1.1.1 Model Formulation

We consider the multicommodity network flow problem (MCNFP) with the following structure: commodities flow over a network such that the aggregate flow on each arc does not exceed some joint capacity. Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ be a graph with a set of vertices $\mathcal{V} = \langle m \rangle$ (where the notation $\langle m \rangle$ represents the set $\{1, 2, \ldots, m\}$) and a set of edges $\mathcal{E} = \{(i,j) | \, i,j \in \mathcal{V}\}$, where $|\mathcal{E}| = n$. Let $\langle K \rangle$ be the set of commodities flowing on $\mathcal{G}$. We will use the following formulation :

[MCNFP]

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & Ax = b \\ & Ex \leq d \\ & 0 \leq x \leq u \end{aligned}$$

where:

f: $\Re^{Kn} \rightarrow \Re$ is the cost function, assumed to be convex and at least twice continuously differentiable

$x \in \Re^{Kn}$ is the vector of arc flows,

E is the $s \times Kn$ coupling constraint matrix,

d is the vector of length s of coupling arc capacities,

A is a block-diagonal matrix of dimension $Km \times Kn$ with component submatrices A_k along the diagonal,

A_k is the node-arc incidence matrix of dimension $m \times n$ for the flows of commodity k on graph $\mathcal{G}$. It is identical for each commodity $k \in \langle K \rangle$,

b is the vector of dimension Km of supplies and demands for each commodity,

u is the vector of dimension Kn of upper bounds on the flows for each commodity on each arc.

The matrix E from the coupling constraints has a generalized upper bounding (GUB) structure:

$$E = \begin{bmatrix} 1 & & & 1 & & & \cdots & 1 & & & \\ & 1 & & & 1 & & & & 1 & & \\ & & \ddots & & & \ddots & & & & \ddots & \\ & & & 1 & & & 1 & \cdots & & & 1 \end{bmatrix}$$

By relaxing the coupling constraint, the problem decomposes into K independent subproblems (one for each commodity) since A has a block-angular structure

$$A = \text{diag}[A_1, A_2, \ldots, A_K],$$

and the vectors x decomposes by commodity:

$$x = [x_1, x_2, \ldots, x_K]^T.$$

Each x_k has dimension $n \times 1$ and is the vector of flows of commodity k. The vectors b and u decompose similarly by commodity.

Transposition is indicated with a superscript ^{T}, gradient vector with the symbol ∇, and second derivative matrix by ∇^2. The interior of a set S is denoted $\text{int}\, S$.

1.1.2 Traditional Approaches to Multicommodity Network Flows

In this section we review methods based on basis partitioning and decomposition methods. These methods have been primarily developed for the linear multicommodity network problem. Basis partitioning methods takes advantage of the underlying network structure of the problem by maintaining part of the simplex basis matrix as a network basis. This allows the use of graph data structures and techniques that have been developed for fast and efficient solution of one-commodity linear network flow problem. These methods have been employed by Kennington [1977], Stone [1988] and a dual variant has been developed by Grigoriadis and White [1972].

Decomposition methods proceed by splitting the solution procedure into two phases: a subproblem phase and master problem phase. A master problem is solved to coordinate the results of subproblems where each subproblem is a minimum cost network flow problem for a single commodity. Two main decomposition algorithms have been studied in the literature: price-directive decomposition and resource-directive decomposition. In price-directive decomposition, the coordination between the master program and the subprograms is achieved through a pricing mechanism which changes the objective functions (prices) of the subprograms. The objective is to obtain a set of prices (dual variables) so that the subprograms yield an optimal solution to the original problem. This principle is at the heart of the Dantzig-Wolfe decomposition. Some of the primary references for this method are Bazaraa and Jarvis [1977], Tomlin [1966] and Wollmer [1972] among others. Recently, due to the development of parallel supercomputers, price-directive decomposition received renewed interest, see for example Ho and Gnanendran [1989] and Wu and Lewis [1989].

An alternative decomposition scheme is the resource-directive decomposition. The idea is to distribute the arc capacity among individual commodities in such a way that solving the K decoupled network subprograms yields an optimal solution to the original problem. At each iteration, a resource allocation is performed and the network subprograms are solved. The sum of the capacities allocated to an arc is less than or equal to the total arc capacity specified in the original problem. Therefore, in this scheme the subprograms produce feasible solutions to the original problem. However, resource directive decomposition is not guaranteed to produce an optimal solution to the multicommodity flow problem. Tangential approximation and subgradient optimization techniques have been used in resource-directive decomposition to solve the nondifferentiable convex master problem, see for instance Kennington [1978], Kennington and Shalaby [1977].

Several solution techniques have also been developed for the nonlinear multicommodity network flow problem. To sample a few, we can cite algorithms based on variations of the Frank-Wolfe method, see for example Leblanc, Morlok and Pierskalla [1975]. However, these methods are known to have mediocre convergence behaviour. Dembo and Tulowitzki [1988] used a second order truncated Newton method to improve the rate of convergence. Bertsekas and Gafni [1982] use a superlinearly convergent projected Newton algorithm to solve nonlinear multicommodity network problems which arise in telecommunications. Meyer and Chen [1988] developed a parallel decomposition method based on a linearization of the

convex nonlinear objective function and trust regions. They report computational results on a multiprocessor parallel computer with very large traffic assignment problems.

Recently, Brown et al. [1989] developed a decomposition method based on the use of penalty and augmented Lagrangian functions. These methods place the arc capacity constraints into the objective function through a penalty term and try to solve the resultant linearly constrained nonlinear problem for increasing values of the penalty parameter. The linear-quadratic penalty method developed in this manuscript is very similar in spirit to the development of these methods. However it presents certain advantages by virtue of the choice of the penalty function. Moreover, parallel decomposition opportunities are not exploited in Brown et al. [1989].

1.1.3 Recent Approaches to Multicommodity Network Flows

In this section we consider recently developed methods for the solution of multicommodity network problems. Some of these ideas were well-known but their use in multicommodity literature appears to be a novelty. Barrier functions and multiplier methods have been studied in depth, see for instance Fiacco and McCormick [1968] and Bertsekas [1982]. Some, however, are new such as the decomposition method based on the notion of coercion functions, Feinberg [1989], Zenios, Qi and Armstrong [1991]. Numerical results reported with these methods seem to be the most promising in the literature so far along with the linear-quadratic penalty method which will be described in subsequent chapters. Therefore a detailed review of these methods here appears to be a worthwhile undertaking.

By placing the mutual capacity constraints into the objective function via a penalty or barrier function, the block angular structure of the constraint matrix can be exploited. This goal can be achieved in a variety of ways. We will discuss here a barrier method by Schultz and Meyer [1990], methods based on Augmented Lagrangians, Brown et al. [1989] and a decomposition method based on the notion of coercion functions, Zenios, Qi and Armstrong [1991]. Before we do so, however, we also want to mention the thesis by Schneur [1991] where an ϵ-scaling algorithm is developed for the solution of multicommodity flow problems and network problems with side constraints.

Methods Based on Barrier Functions.

In this section, we present a decomposition method based on barrier functions for solving block angular linear programs due to Schultz and Meyer [1990].

The algorithm begins by finding x^0 as the solution of the relaxed problem

[RMNF]

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & Ax = b \\ & 0 \leq x \leq u \end{aligned}$$

A two-phase barrier method, described next, starts with x^0 and produces a solution that is optimal within a specifed tolerance.

Define the feasible set of the network constraints as

$$B = \{x | Ax = b \text{ and } 0 \leq x \leq u\},$$

and the feasible set of capacity constraints

$$C = \{x | Ex \leq d\}.$$

Suppose x^i is given. At each iteration, the original problem is approximated by the *barrier problem*

$$\min_{x \in B} \phi(x, \tau, \theta) = f(x) - \tau \sum_{j=1}^{s} \ln(\theta_j - E_j x) \tag{1.1}$$

where $0 < \tau \in \Re$ and $0 < \theta \in \Re^s$ are parameters and ln is the natural logarithm. Note that $\phi(x, \tau, \theta)$ is convex in x with domain dom $\phi(., \tau, \theta) = \{x | Ex < d\}$. Allowing $\theta \neq d$ has the property of shifting the barrier. For this reason, the penalty portion of ϕ is called a *shifted logarithmic barrier function.*

After finding x^0 as the solution of [RMNF] the parameters θ^0 and τ^0 are specified by taking $\tau^0 > 0$ and

$$\theta_j^0 = \begin{cases} d_j & \text{if } E_j x^0 < d_j \\ E_j x^0 + \Theta & \text{if } E_j x^0 \geq d_j \end{cases}$$

where $\Theta \geq 0$. This will have the effect of making $x_0 \in B$ an interior point of dom $\phi(., \tau, \theta)$. Then x^{i+1} is computed by doing one step of a multi-dimensional search on the barrier

problem (1.1). If $Ex^{i+1} < d$, a feasible interior point has been obtained. If not, then θ is modified by taking

$$\theta_j^{i+1} = \begin{cases} d_j & \text{if } E_j x^{i+1} < d_j \\ \lambda_\theta E_j x^{i+1} + (1-\lambda_\theta)\theta_j^i & \text{if } E_j x^{i+1} \geq d_j \end{cases}$$

where $\lambda \in (0,1)$ is a constant, while maintaining $\tau^{i+1} = \tau^i$. Then set $i \leftarrow i+1$ and repeat the process.

Schultz and Meyer [1990] have shown that if a point $x \in B \cap \text{int}C$ exists, then such a point may be found in a finite number of iterations. In order to achieve this, however, the barrier problems (1.1) must be solved accurately.

Once the method has $\theta^{\hat{i}} = d$ so that $x^{\hat{i}} \in B \cap \text{int}C$, $\theta^i = d$ is maintained and the effect of barrier terms in ϕ is gradually relaxed. This is accomplished by letting $\tau^i \downarrow 0$. Convergence results for barrier methods, see Fiacco and McCormick [1968], state that minimizers of (1.1) converge to minimizers of the original problem as $\tau \downarrow 0$. In Schultz and Meyer [1990], a sequence $\{\tau^i\}$ generated by the recurrence

$$\tau^{i+1} \leftarrow \max\{\lambda_\tau \tau^i, \tau_{\text{inf}}\}$$

is used, where $\tau^0, \tau_{\text{inf}} > 0$ and $\lambda_\tau \in [0,1)$. The following result can then be used to choose a suitable τ_{inf}.

Theorem 1 *Let x^* be an optimal solution of [MCNFP], and say $\bar{x}$ is a minimizer of (1.1) with $\tau > 0$ and $\theta = d$. Then $f(\bar{x}) \geq f(x^*) \geq f(\bar{x}) - \tau s$. (Recall that s is the number of rows in E.)*

A proof of the result can be found in Fiacco and McCormick [1968, p. 102] or in McCormick [1983, p.341]. So if the user chooses $\tau_{\text{inf}} = \varepsilon/s$ then solving the barrier problem (1.1) in the limit produces limit points $\bar{x} \in B \cap \text{int}C$ such that $|f(\bar{x}) - f(x^*)| \leq \varepsilon$.

In summary, this method is a three phase process. The first phase may be called the *relaxed phase.* It consists of solving the problem [RMNF]. If this problem is infeasible, so is the original problem. The second phase may be called the *feasibility phase.* During this phase a point $x^{\hat{i}} \in B \cap \text{int}C$ is obtained by forcing $\theta = d$. If the feasibility phase succeeds, one moves to the final phase where one seeks to approximate an optimal solution of [MCNFP] by reducing the effects of the barrier term, i.e. by reducing τ. Very encouraging computational results with a set of large multicommodity network flow problems are

reported. The same problems were used in our study and more will be said about them in subsequent chapters.

Multiplier Methods for Multicommodity Flows.

In this section we consider a related class of methods usually referred to as *method of multipliers*. A primary reference in this subject is the text by Bertsekas [1982].

Consider the nonlinear program

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{subject to} & \\ & g(x) = 0 \end{array}$$

where f is a real valued function and $g(x) = 0$ is a system of m equality constraints. The augmented Lagrangian penalty problem for the above problem is

$$\min_x L(x, \mu, y) = \min_x f(x) + yg(x) + \frac{\mu}{2}\|g(x)\|_2^2 \tag{1.2}$$

where $y \in \Re^m$ and $\mu > 0$ is a scalar. A typical iteration of the method of multipliers would be to solve for given μ^ℓ and y^ℓ

$$\min_x L(x, \mu^\ell, y^\ell) = \min_x f(x) + y^\ell g(x) + \frac{\mu^\ell}{2}\|g(x)\|_2^2 \tag{1.3}$$

followed by updates of the multiplier vector according to

$$y^{\ell+1} = y^\ell + \mu^\ell g(\underline{x}) \tag{1.4}$$

where $\underline{x}$ is the minimizer of $L(x, \mu^\ell, y^\ell)$ in (1.3). Necessary modifications to handle inequality constraints are given in Bertsekas [1982]. Using this conversion, we consider, for a positive penalty multiplier $\mu > 0$, the augmented Lagrangian corresponding to problem [MCNFP]

$$L_\mu(x, y) = f(x) + \frac{1}{2\mu}\sum_{i=1}^{r}[(\max(0, y^i + \mu(E_i x - d_i)))^2 - (y^i)^2]$$

where y^i denotes the ith coordinate of the vector $y \in \Re^s$. The method of multipliers we consider consists of the minimization step

$$x_k = \arg\min_{x \in X} L_\mu(x, y_k) \tag{1.5}$$

where

$$X = \{x | Ax = b \text{ and } 0 \leq x \leq u\},$$

and the multiplier update

$$y_k^i = \max(0, y_k^i + \mu_k(e_i x - s_i)) \quad i = 1, \ldots, s \tag{1.6}$$

The starting vector y_0 is arbitrary and $\{\mu_k\}$ is a non–decreasing sequence of positive numbers. We note that an exterior penalty method with the quadratic penalty function

$$\phi(t) = (\max\{0, t\})^2 \tag{1.7}$$

can be seen as a special case of the method of multipliers without the multiplier update (1.6). The method of multipliers is known to possess some theoretical advantages over exterior penalty methods based on the quadratic penalty function (1.7). This has also been supported by computational testing, see Bertsekas [1982]. Furthermore, the elimination of a subset of constraints gives one the incentive to consider this method as a candidate for the solution of multicommodity network flow problem. As is shown in Bertsekas [1982], convergence in the method of multipliers can usually be attained without the need to increase the penalty multiplier μ to infinity as is the case with sequential exterior penalty methods, thereby alleviating the ill-conditioning associated with penalty methods. In addition, the multiplier iteration (1.6) converges to a Lagrange multiplier vector of the original problem as defined in Rockafellar [1970].

As in the case of the linear-quadratic penalties, the nonseparability constitutes an important issue in the minimization of the augmented Lagrangian. Linearization methods can also be applied in this context to induce separability and to exploit the block angular structure of the multicommodity flow matrix. This approach has been proposed in the Ph.D. thesis by Liu [1988] and tested on large scale multicommodity network problems by Brown et al. [1989]. A linearization scheme allows the decomposition of the augmented Lagrangian minimization step (1.5) into independent subproblems for each commodity that can be solved in parallel. Brown et al. [1989] report promising computational results with a set of large multicommodity ammunition distribution problems. No computational testing on parallel architectures is available with these methods.

Methods Based on Coercion Functions.

The coercion function method was first proposed by Feinberg [1989] for linear programs. It was later extended by Zenios, Qi and Armstrong [1991] for linearly constrained nonlinear programs. Similar to the classical Dantzig-Wolfe decomposition algorithm, the coercion function method decomposes a multicommodity network flow problem into a series of master and subproblems. In the classical Dantzig-Wolfe decomposition, the subproblems generate proposals which are feasible to the overall problem and the master problem generates *prices* for these proposals. However, the master problem must dictate the most recent optimal dual solution to each of the subproblems. Unlike the Dantzig-Wolfe decomposition, the master problem does not dictate the optimal dual solution to each of the subproblems. Instead, a coercion function is added to the objective function of the subproblems to coerce the subproblems to generate the optimal solution to the original problem in the limit. The information about the newly generated optimal solution to the master problem is passed to the subproblems through a parameter vector α. At each iteration, the master problem uses subproblem proposals to generate new values for α and pass it to the subproblem. And the subproblem will generate new feasible solutions which converge in the limit to the optimal solution of the original problem.

We now introduce the notion of coercion function and describe the decomposition algorithm.

Subproblem Formulation

The dual problem corresponding to [**MCNFP**] can be formulated as :

[**MCNFD**] :

$$\underset{\phi,\pi,\overline{\gamma},\underline{\gamma}}{\text{Maximize}} \left\{ \begin{array}{l} \text{Minimize} \\ x \in R^{Kn} \\ \text{Subject to :} \end{array} \right. \begin{array}{l} [F(x) + \phi(Ex - d) + \pi(Ax - b) + \overline{\gamma}(x - u) - \underline{\gamma}x] \} \\ \phi \in R^{n}_{+} \\ \overline{\gamma},\underline{\gamma} \in R^{Kn}_{+} \end{array}$$

where $\phi, \pi, \overline{\gamma}, \underline{\gamma}$ are vectors of dual variables corresponding to constraints $Ax = b, Ex \leq d, x \leq u$ and $-x \leq 0$ in [**MCNFP**]. We use x^* and ϕ^* to denote the optimal solutions of the primal and the dual nonlinear programs.

The coercion function decomposition method consists of two phases: the subproblem phase and the master problem phase. The subproblem constraint set is composed of the network flow balance constraints $Ax = b$ and the bounds $0 \leq x \leq u$. The subproblem can be posed as follows:

[SP]

$$\begin{array}{ll} \underset{x}{\text{Minimize}} & F(x) + f(\alpha, x) \\ \text{Subject to :} & Ax = b \\ & 0 \leq x \leq u \end{array}$$

where $f(\alpha, x)$ is the coercion function we added to the subproblem objective. If the objective function for the original multicommodity network flow problem $F(x)$ is block-separable by commodity, that is $F(x) = \sum_{k=1}^{K} F_k(x_k)$, and the coercion function $f(\alpha, x)$ is chosen so as to retain block-separability, that is $f(\alpha, x) = \sum_{k=1}^{K} f_k(\alpha, x_k)$, where x^k is the flow for commodity k, then the subproblem can be decomposed into K independent subproblems. For $k = 1, 2, \ldots, K$:

[SP_k]

$$\begin{array}{ll} \underset{y_k}{\text{Minimize}} & F_k(y_k) + f_k(\alpha, y_k) \\ \text{Subject to :} & A_k y_k = b_k \\ & 0 \leq y_k \leq u_k \end{array}$$

A nonlinear function $f(\alpha, x)$ is called a *coercion function* with respect to [MCNFP] if there exists some α^* such that x^* minimizes the subproblem, where x^* is the optimal primal solution for the original problem. That is, if the subproblem is solved with $\alpha = \alpha^*$, it will return x^* as its optimal solution. The following result characterizes precisely the coercion functions.

Theorem 2 *Let x^* be an optimal solution to the original problem. If $f(\alpha, x)$ is strictly convex in x for all α and $\nabla_x f(\alpha^*, x^*) = \phi^* A$, then $f(\alpha, x)$ is a coercion function. That is, given $\alpha = \alpha^*$ the optimal solution to the subproblem is x^*.*

See Zenios, Qi and Armstrong [1991] for details.

An example of coercion function is the quadratic function used by Feinberg [1989]. It is particularly attractive for multicommodity network flow problems since it separates by commodity :

$$f(\alpha, x) = \frac{1}{2}\|x - \alpha\|^2$$

$$\|x - \alpha\|^2 = \|x_1 - \alpha_1\|^2 + \|x_2 - \alpha_2\|^2 + \ldots + \|x_K - \alpha_K\|^2$$

Calculating α for this function is also simple. Since $\nabla_x f(\alpha, x) = \nabla_x \frac{1}{2}\|x - \alpha\|^2 = x - \alpha$, from equation $x - \alpha = \phi A$, we have $\alpha = x - \phi A$.

Master Problem Formulation

The master problem phase can be seen as a coordination phase which gives the optimal solution for the overall multicommodity network flow problem based on the convex combination of the subproblem proposals. Let S denote the feasible set for the subproblems, $S = \{x | Ax = b, 0 \leq x \leq u\}$, then the original problem can be expressed as

[MP]

$$\begin{array}{lll} \text{Minimize} & & F(x) \\ x & & \\ \text{Subject to} & & Ex \leq d \\ & & x \in S \end{array}$$

Since S is a polyhedral convex set, any convex combination of points in S is still in S (i.e. If $y_i \in S$ for $i = 0, 1, \ldots, l-1$, and $x = \sum_{i=0}^{l-1} \lambda_i y_i$ such that $\lambda_i \geq 0$ and $\sum_{i=0}^{l-1} \lambda_i = 1$, then $x \in S$). Based upon the above arguments, given proposals $y_i \in S$ for $i = 0, 1, \ldots, l-1$ from subproblems, only a convex combination of these proposals can be considered. Therefore the original problem can be reformulated as follows:

[MP]

$$\begin{aligned} \text{Minimize} \quad & F\Big(\sum_{i=0}^{l-1}\lambda_i y_i\Big) \\ x \quad & \\ \text{Subject to :} \quad & E\sum_{i=0}^{l-1}\lambda_i y_i \le d \\ & \sum_{i=0}^{l-1}\lambda_i = 1 \\ & \lambda_i \ge 0 \end{aligned}$$

It is known from theorem 2 that given x^* and ϕ^* we can find α^* by solving $\nabla_x f(\alpha^*, x^*) = \phi^* A$. But if x^* was known, the solution of the problem would not be attempted at all. Therefore the following iterative procedure is used to find the optimal x^*, ϕ^* and α^* based on the relation $\nabla_x f(\alpha, x) = \phi A$. Suppose l subproblem proposals y_i for $i = 0, 1, \ldots, l-1$ given, solving the master problem one gets $x = \sum_{i=0}^{l-1}\lambda_i y_i$, the primal optimal solution and ϕ, the dual optimal solutions corresponding to the coupling constraints set $E\sum_{i=0}^{l-1}\lambda_i y_i \le d$.

The parameter α is computed such that $\nabla_x f(\alpha, x) = \phi A$ and passed to the subproblem. Then the subproblem generates a new proposal y_l and presents it to the master problem. It can be shown that the new proposal generated by the subproblem will always provide a descent direction for the master problem. Hence in the subsequent iteration the solution to the master problem will be improved. For more details the reader should consult Zenios, Qi and Armstrong [1991].

Chapter 2

A Quadratic Smoothing of the 1-Norm Exact Penalty Function for Convex Constrained Optimization

2.1 Introduction

In this chapter we develop a smoothing technique for the 1-norm exact penalty function for convex constrained optimization. We analyze the exactness properties of the smooth penalty function. In subsequent chapters, the smooth penalty function is used to eliminate the non-network (side) constraints from the constraint set and thus to obtain an optimization problem with network flow conservation constraints. This chapter paves the way to the analytical development of this methodolgy in a general setting.

The idea of using penalty function methods to simplify optimization problems is probably as old as the field of nonlinear programming itself. Starting with the work of Fiacco and McCormick [1968] on penalty functions, attempts were made to solve general purpose nonlinear programming problems by reducing the problem to a sequence of problems with either no constraints or with simple constraints. Improvements on this work came later in the form of Augmented Lagrangian and exact penalty functions which were explored by a number of

authors (for a complete account of the subject refer to the book by Bertsekas [1982]).

It is well known that a solution to a convex program can be obtained by solving the exact penalty problem for a certain value of the penalty parameter provided that it is larger than a threshold value characterized by the largest, in magnitude, of the Lagrange multipliers of the original problem, Bertsekas [1975]. However, this assertion is equivalent to a nondifferentiability requirement on the penalty function as shown in Bertsekas [1975], which rules out a straightforward application of gradient based descent methods for the solution of the penalty problem. On the other hand, it is possible to obtain a Lagrange multiplier to the original problem in a finite number of minimizations even with a differentiable penalty function provided the penalty function is "steep" enough and some special structure is present in the original problem. These conditions are made precise in Bertsekas [1975] and illustrated with the quadratic penalty function. It is shown that for polyhedral convex programs, for a certain threshold value of the penalty parameter, it is possible to obtain a Lagrange multiplier vector to the original problem provided that one exists. But with the quadratic penalty function, the characterization of the threshold value in terms of the Lagrange multipliers is lost.

This chapter contains the following results. First we examine the properties of the ϵ-smooth penalty function as an approximation to the ℓ_1 exact penalty function. We show that an a priori upper bound to the approximation error can be computed as a function of the penalty parameters. We prove that although an optimum point to the original problem is not necessarily an optimum of the penalty problem for values of the penalty parameter larger than the threshold, its suboptimality can be bounded by a computable constant. This result is a partial generalization of the classical result, see for instance Charalambous [1980], which states that if the original problem is solvable, then the penalty problem is solvable and that the optimal solutions coincide provided that the penalty parameter is larger than a threshold. Then we show that for an ϵ-smoothing of the nondifferentiable ℓ_1 penalty function it is possible to obtain a feasible solution of an $O(\epsilon)$ perturbation of the original problem, i.e., the maximum degree of infeasibility in any constraint is bounded by a term linear in ϵ provided that the penalty parameter is larger than a threshold value characterized in terms of the Lagrange multipliers to the perturbed problem. We also show that the solution to the penalty problem is such that it has at least one constraint ϵ-feasible or satisfied if the penalty parameter is specified larger than the threshold. The main result of

the chapter is that it is possible to compute a ϵ feasible solution if the penalty parameter is specified as a constant multiple of the threshold value for the nondifferentiable case. This result removes the ϵ dependency of the threshold value for ϵ exactness as proved in Truemper [1975] for the quadratic penalty function. It is also shown that due to the differentiability property the solution to the smooth penalty problem yields the Lagrange multiplier vector to a linear program assuming it is unique and provided that the penalty parameter is specified larger than the Lagrange multiplier vector. Therefore it is possible to compute the Lagrange multiplier to a linear program in a finite number of minimizations. Thus we recover the precise characterization of the threshold value, which was well known for the nondifferentiable case, with a differentiable penalty function. Assuming at least once continuous differentiability of the problem functions, the penalty minimizations can be carried out using any gradient based descent method. The rest of this chapter is devoted to making these ideas precise.

2.2 Preliminaries

We consider the problem

[NLP]

$$\begin{array}{ll} \underset{x \in X}{\text{minimize}} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0 \ \forall \ \ i = 1, \ldots, K \end{array}$$

where the functions $f_i : \Re^n \longmapsto \Re$ are convex and at least once continuously differentiable and X is a convex and compact subset of $\Re^n$. We further assume that

A1. [NLP] has a non-empty and compact optimal solution set and,

A2. [NLP] has at least one Lagrange multiplier vector.

Consider the penalty function $p : \Re \mapsto \Re$ such that

$$p(t) \begin{cases} > 0 & \text{for } t > 0 \\ = 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where t is a scalar variable. We want to obtain a solution to [NLP] by solving the following penalty problem

[EP]

$$\min_{x \in X} f_0(x) + \sum_{i=1}^{K} p(f_i(x)) \tag{2.2}$$

In this document we will focus on the ℓ_1 penalty function defined as

$$p(t) = \mu \max\{0, t\}. \tag{2.3}$$

where μ is a positive scalar which determines the slope of the function and the severity of the penalty. It is known that for a threshold value of the penalty parameter μ characterized in terms of some Lagrange multiplier vector of the original problem, the solutions to [EP] and [NLP] coincide, Bertsekas [1975].

2.2.1 A Smooth Approximation

The function p defined by (2.3) is not continuously differentiable at $t = 0$. In order to gain access to gradient based minimization techniques to solve [EP] we consider an ϵ-smoothing of the function p. Let the ϵ-smoothed function $\tilde{p}$ be the following:

$$\tilde{p}(t) = \begin{cases} 0 & \text{if } t \leq 0 \\ \mu \frac{t^2}{2\epsilon} & \text{if } 0 \leq t \leq \epsilon \\ \mu(t - \frac{\epsilon}{2}) & \text{if } t \geq \epsilon \end{cases} \tag{2.4}$$

where ϵ is a positive scalar, see Figure 2.1. We note that the smooth penalty function introduced here can be obtained as a special case of the smoothing technique introduced in Bertsekas [1973]. A similar technique was used in Zang [1980] for min-max problems extending the work in Bertsekas [1973]. The smoothing technique introduced in Zang [1980] is symmetric around the kink while our smoothing approximation is asymmetric. The following can be easily verified:

$$\lim_{\epsilon \to 0} \tilde{p}(t) = p(t) \tag{2.5}$$

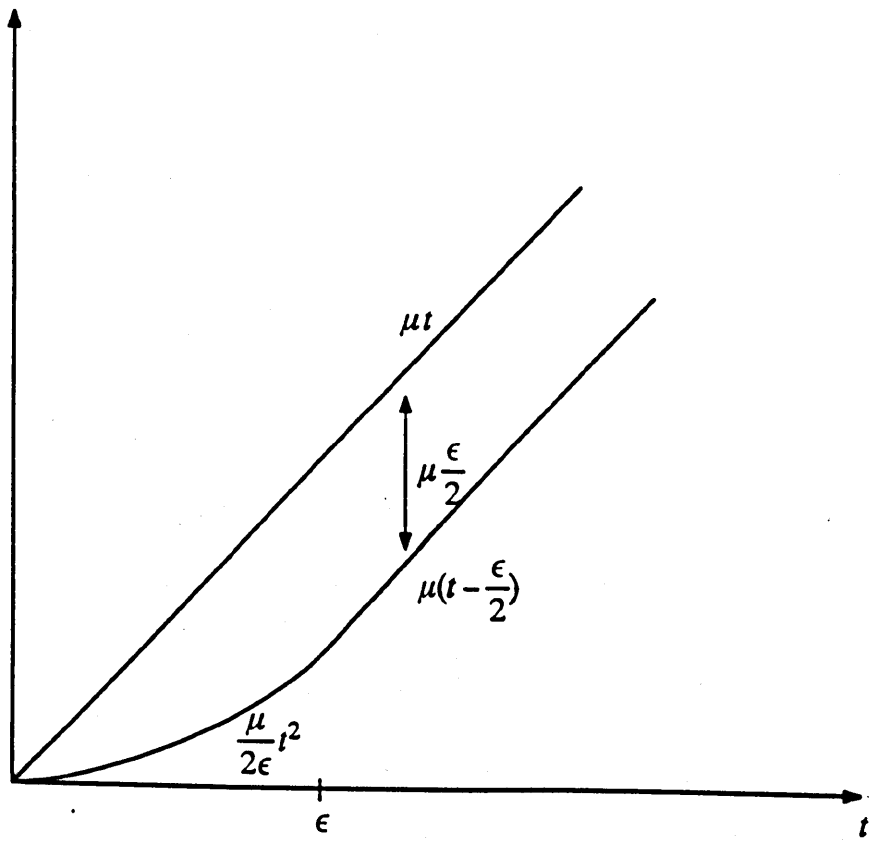


Figure 2.1: Linear-quadratic smooth penalty function.

Let us define

$$F(x,\mu) = f_0(x) + \sum_{i=1}^{K} p(f_i(x)) \tag{2.6}$$

and

$$\tilde{F}(x,\mu,\epsilon) = f_0(x) + \sum_{i=1}^{K} \tilde{p}(f_i(x)) \tag{2.7}$$

We can now state the following result which gives an a priori upper bound on the smoothing approximation error.

Lemma 1. Let the functions F and $\tilde{F}$ be as defined by (2.6) and (2.7). Then

$$0 \leq F(x,\mu) - \tilde{F}(x,\mu,\epsilon) \leq K\mu\frac{\epsilon}{2} \tag{2.8}$$

for any $x \in \Re^n$, $\mu > 0$ and $\epsilon \geq 0$.

Proof. Using the definitions of p and $\tilde{p}$, we get

$$0 \leq p(f_i(x)) - \tilde{p}(f_i(x)) \leq \mu\frac{\epsilon}{2} \qquad \forall\ i = 1,\ldots,K \quad \text{and} \quad x \in \Re^n \tag{2.9}$$

Adding up for all i and multiplying with $\mu > 0$ we obtain

$$0 \leq \sum_{i=1}^{K} p(f_i(x)) - \sum_{i=1}^{K} \tilde{p}(f_i(x)) \leq K\mu\frac{\epsilon}{2} \qquad \forall\ i = 1,\ldots,K \quad \text{and} \quad x \in \Re^n \tag{2.10}$$

Then the result immediately follows from the definitions of F and $\tilde{F}$. ∎

We propose to solve the following smoothed penalty problem

[SEP]

$$\min_{x \in X} \tilde{F}(x,\mu,\epsilon) \tag{2.11}$$

instead of solving **[EP]**. Before we proceed we need the following definition.

Definition 1. A vector $\tilde{x}$ is ϵ-feasible if

$$f_i(\tilde{x}) \leq \epsilon \qquad \forall\ i = 1,\ldots,K.$$

Definition 2. A constraint i is ϵ-violated at $\tilde{x}$ if

$$f_i(\tilde{x}) > \epsilon$$

Now we can prove the following result which gives an a priori upper bound on the error incurred by solving the smooth penalty problem [SEP] in lieu of the nondifferentiable penalty problem [EP].

Proposition 2. Let x^* be an optimal solution of [EP] and $\tilde{x}$ be an optimal solution of [SEP] for some μ and ϵ. Then

$$0 \leq F(x^*, \mu) - \tilde{F}(\tilde{x}, \mu, \epsilon) \leq K\mu\frac{\epsilon}{2}. \tag{2.12}$$

Proof. Using (2.8), we can immediately rewrite the right inequality as

$$F(x, \mu) \leq \tilde{F}(x, \mu, \epsilon) + K\mu\frac{\epsilon}{2}. \tag{2.13}$$

Taking the infimum, we obtain

$$\inf_{x \in X} F(x, \mu) \leq \inf_{x \in X} \tilde{F}(x, \mu, \epsilon) + K\mu\frac{\epsilon}{2}. \tag{2.14}$$

which proves the right hand inequality. The left hand inequality can be proved similarly and is omitted. ∎

An immediate consequence of Proposition 2 can be obtained if in addition to the assumptions stated in the proposition, we assume that x^* is a feasible thus optimal solution to [NLP] and $\tilde{x}$ is ϵ-feasible for the original problem.

Proposition 3. Let x^* be an optimal solution of [EP] and $\tilde{x}$ be an optimal solution of [SEP] for some μ and ϵ. Furthermore let x^* be a feasible solution to [NLP] and $\tilde{x}$ be ϵ-feasible. Then

$$0 \leq f_0(x^*) - f_0(\tilde{x}) \leq K\mu\epsilon \tag{2.15}$$

Proof. By hypothesis, $\tilde{x}$ is ϵ-feasible which implies the following

$$\sum_{i=1}^{K} \tilde{p}(f_i(\tilde{x})) \leq K\mu\frac{\epsilon}{2}. \tag{2.16}$$

Again by hypothesis, x^* is a solution to [**NLP**], which implies that

$$\sum_{i=1}^{K} p(f_i(x^*)) = 0. \tag{2.17}$$

From Proposition 2, we have

$$0 \leq f_0(x^*) + \sum_{i=1}^{K} p(f_i(x^*)) - f_0(\tilde{x}) - \sum_{i=1}^{K} \tilde{p}(f_i(x)) \leq K\mu\frac{\epsilon}{2} \tag{2.18}$$

Substituting (2.16) and (2.17) into (2.18) and rearranging terms the result is established. ∎

The above assertion does not specify conditions for the penalty parameter μ. It is known, Bertsekas [1975], that for μ larger than the largest of Lagrange multipliers of the original problem [**NLP**], an optimal solution x^* to [**NLP**] is also an optimal solution for [**EP**]. This result provided the motivation for designing exact penalty methods, Bertsekas [1982]. The question we address next is how this result is affected by the ϵ-smoothing of the exact penalty function? The answer, as we show, is that although an optimal solution to [**NLP**] does not necessarily coincide with an optimal solution to [**SEP**], its suboptimality can be bounded as a function of the penalty and smoothing parameters. We proceed with a precise statement and proof of this result.

Proposition 4. Let (x^*, y^*) be a primal-dual optimal pair for [**NLP**]. Then for some $\epsilon \geq 0$

$$\tilde{F}(x^*, \mu, \epsilon) \leq \tilde{F}(x, \mu, \epsilon) + K\mu\frac{\epsilon}{2} \quad \forall \quad x \in X \tag{2.19}$$

provided that $\mu > y_i^*$ for $i = 1, \ldots, K$.

Proof. For simplicity of exposition, we assume $X = \Re^n$ and define

$$f_i^+(x) = \max\{0, f_i(x)\}$$

By convexity of f_i for $i = 0, \ldots, K$ we have for a fixed but arbitrary $x \in \Re^n$

$$f_i(x) \geq f_i(x^*) + \nabla f_i(x^*)^T(x - x^*) \tag{2.20}$$

Since (x^*, y^*) is a primal-dual optimal pair, by the first-order optimality conditions we have the following:

$$\nabla f_0(x^*) = -\sum_{i=1}^{K} y_i^* \nabla f_i(x^*) \tag{2.21}$$

$$y_i^* f_i(x^*) = 0 \quad \forall \quad i = 1, \ldots, K \tag{2.22}$$

$$y_i^* \geq 0 \quad \forall \quad i = 1, \ldots, K \tag{2.23}$$

Using (2.20)-(2.21)-(2.22) and the definition of F, we obtain

$$\begin{aligned}
F(x,\mu) &\geq f_0(x^*) + \nabla f_0(x^*)^T(x - x^*) + \mu \sum_{i=1}^{K} f_i^+(x) \\
&= f_0(x^*) - \sum_{i=1}^{K} y_i^* \nabla f_i(x^*)^T(x - x^*) + \mu \sum_{i=1}^{K} f_i^+(x) \quad \text{(using (2.21))} \\
&\geq f_0(x^*) - \sum_{i=1}^{K} y_i^* (f_i(x) - f_i(x^*)) + \mu \sum_{i=1}^{K} f_i^+(x) \quad \text{(using (2.20))} \\
&= f_0(x^*) - \sum_{i=1}^{K} y_i^* f_i(x) + \mu \sum_{i=1}^{K} f_i^+(x) \quad \text{(using (2.22))}
\end{aligned}$$

Since $f_i(x) \leq f_i^+(x)$ we have

$$F(x,\mu) \geq f_0(x^*) + \sum_{i=1}^{K} (\mu - y_i^*) f_i^+(x)$$

Thus, for $\mu \geq y_i^*$ for $i = 1, \ldots, K$, we get

$$F(x,\mu) \geq f_0(x^*) \tag{2.24}$$

However, from Lemma 1, we have

$$F(x,\mu) - \tilde{F}(x,\mu,\epsilon) \leq K\mu\frac{\epsilon}{2} \tag{2.25}$$

Rewriting (2.24) as

$$f_0(x^*) - F(x,\mu) \leq 0 \tag{2.26}$$

we observe that since x^* is also feasible, then

$$f_0(x^*) = \tilde{F}(x^*, \mu, \epsilon)$$

Thus adding up inequalities (2.25) and (2.26) we obtain

$$\tilde{F}(x^*, \mu, \epsilon) - \tilde{F}(x, \mu, \epsilon) \leq K\mu\frac{\epsilon}{2} \tag{2.27}$$

which establishes the result. ∎

Therefore the suboptimality of x^* in [SEP] is bounded by the quantity $K\mu\frac{\epsilon}{2}$. It can be observed that as ϵ vanishes one recovers the classical assertion that an optimal solution to the original problem coincides with an optimal solution to the penalty problem for a finite value of the penalty parameter μ. Thus, Proposition 4 partially generalizes Proposition 1 of Bertsekas [1975] in this respect. An immediate corollary of Proposition 4 can be stated as follows.

Corollary 5. Let $\bar{x}$ be a minimum point of

$$\min_{x \in X} \tilde{F}(x, \mu, \epsilon)$$

and x^* be an optimal point for the original problem [NLP]. Then

$$f_0(x^*) - \tilde{F}(\bar{x}, \mu, \epsilon) \leq K\mu\frac{\epsilon}{2} \tag{2.28}$$

provided that $\mu > y_i^*$ for $i = 1, \ldots, K$.

Proof. Immediate from Proposition 4. ∎

Proposition 4 also engenders an important consequence together with the following result which is identical to Proposition 3.5 of Zang [1980].

Proposition 6. Let $\{\epsilon_k\} \to 0$ be a sequence of positive numbers and assume that x_k is a solution to

$$\min_{x \in X} \tilde{F}(x, \mu, \epsilon_k) \tag{2.29}$$

for some $\mu > 0$. Also let $\bar{x}$ be an accumulation point of the sequence $\{x_k\}$, then

$$F(\bar{x}, \mu) = F(x(\mu), \mu) \tag{2.30}$$

where $x(\mu)$ is an optimal solution to

$$\min_{x \in X} F(x, \mu).$$

Proof. The result follows directly from the continuity of F, (2.5), and (2.12). ∎

Therefore if the penalty parameter μ in Proposition 5 was specified larger than the threshold value μ^*, the solutions to [EP] and [NLP] would coincide. Hence, we could hope to obtain a solution to the original problem [NLP] in the limit by solving smooth penalty problems for decreasing smoothing parameters ϵ_k and increasing penalty parameters μ_k since the threshold value μ^* is not known in practice. Also, driving ϵ to 0 is not desirable since we recover the nondifferentiable ℓ_1 penalty function. On the other hand, we lose the "exactness" property by solving the smooth penalty problem instead of the nondifferentiable penalty problem. But we will show in the next section that an optimal solution to the smooth penalty problem will violate the constraints at most by a quantity which is a linear function of ϵ if the penalty parameter μ is larger componentwise than some Lagrange multiplier vector of the perturbed problem componentwise. This result is developed in the next section. The main result of the chapter is also given in the next section. We will show that ϵ-feasibility is attained if the penalty problem [SEP] is solved using a penalty parameter value equal to a constant multiple of the threshold value for the nondifferentiable ℓ_1 penalty function.

2.3 Approximate or ϵ-Exactness

In this section we investigate the approximate *exactness* properties that the smooth penalty function inherits from its nondifferentiable counterpart. We are particularly interested in achieving ϵ-feasibility which is an important property. We begin with some observations regarding the consequences of ϵ-feasibility.

Remark 1. Any primal-dual pair $(\tilde{x}, \tilde{y})$ where $\tilde{x}$ is a solution to [SEP] for some ϵ and Lagrange multiplier estimate $\tilde{y}$ is obtained from

$$\tilde{y}_i = \frac{d\tilde{p}}{dt}(f_i(\tilde{x})) \qquad \forall \quad i = 1, \ldots, K. \tag{2.31}$$

satisfies the first order optimality conditions with the exception of the complementary slackness condition. However, it can be easily shown that the error in complementary slackness is bounded by the quantity $\mu\epsilon$. To see this, observe that all constraints are satisfied to within ϵ and that the Lagrange multiplier estimates are given by

$$\tilde{y}_i = \frac{\mu \max(0, \tilde{u}_i)}{\epsilon} \qquad \forall \quad i = 1, \ldots, K.$$

where

$$\tilde{u}_i = f_i(\tilde{x})$$

Therefore we have

$$\tilde{y}_i \tilde{u}_i = \max\{0, \mu \frac{\tilde{u}_i^2}{\epsilon}\} \quad \forall \quad i = 1, \ldots, K$$

However since $\tilde{u}_i \leq \epsilon$ for $i = 1, \ldots, K$ the following holds

$$\tilde{y}_i \tilde{u}_i \leq \mu\epsilon \quad \forall \quad i = 1, \ldots, K \tag{2.32}$$

Hence, the assertion is verified.

Remark 2. For ϵ small enough, the pair $(\tilde{x}, \tilde{y})$ *almost* satisfies the first order optimality conditions by virtue of Remark 1 above. Applying the Standard Implicit Function Theorem to the system of nonlinear equations

$$h(x, y) - z = 0$$

where $h_i(x, y) = y_i f_i(x)$ for $i = 1, \ldots, K$ and $(x^*, y^*, 0)$ is assumed to be a solution, it can be shown that that there exists a $\delta > 0$ such that

$$\|x^* - \tilde{x}\| \leq \delta,$$

$$\|y^* - \tilde{y}\| \leq \delta$$

for all z such that $|z| \leq \mu\epsilon$ by virtue of the complementary slackness error.

The above remarks motivate the effort to compute an ϵ-feasible solution to the original problem by solving the smooth penalty problem [SEP]. We will frequently refer to ϵ or δ-perturbation of problem [NLP] in the sequel. We allude to the following problem:

$$\begin{array}{ll} \underset{x \in X}{\text{minimize}} & f_0(x) \\ \text{subject to} & f_i(x) \leq \delta \ \forall \ \ i = 1, \ldots, K \end{array}$$

where δ is a positive scalar. We begin by stating a result which gives a partial characterization of an optimal solution to the smooth penalty problem with regard to ϵ-feasibility.

Proposition 7. Let $\tilde{x}$ be an optimal solution to

$$\min_{x \in X} \tilde{F}(x, \mu, \epsilon)$$

Suppose also that $(\bar{x}, \bar{y})$ is a primal-dual optimal pair for a ϵ perturbation to [NLP]. Then there is at least one $i \in I = \{1, \ldots, K\}$ such that

$$f_i(\tilde{x}) \leq \epsilon$$

provided that

$$\mu > \bar{y}_i \ \ \forall \ \ i = 1, \ldots, K \tag{2.33}$$

Proof. We will argue by contradiction. Assume $\tilde{x}$ is a point such that all constraints are violated beyond ϵ at $\tilde{x}$. By simple algebraic manipulation,

$$f_0(\tilde{x}) + \mu \sum_{i=1}^{K} (f_i(\tilde{x}) - \frac{\epsilon}{2}) = f_0(\tilde{x}) + \mu \sum_{i=1}^{K} (f_i(\tilde{x}) - \epsilon) + K\mu\frac{\epsilon}{2} \tag{2.34}$$

Using (2.33), the definition of a Lagrange multiplier Rockafellar [1970], and the ϵ-feasibility of $\bar{x}$ we have the following

$$f_0(\tilde{x}) + \mu \sum_{i=1}^{K} (f_i(\tilde{x}) - \epsilon) + K\mu\frac{\epsilon}{2}$$

$$
\begin{aligned}
&> f_0(\tilde{x}) + \sum_{i=1}^{K} \bar{y}_i (f_i(\tilde{x}) - \epsilon) + K\mu\frac{\epsilon}{2} \\
&\geq f_0(\bar{x}) + \sum_{i=1}^{K} \bar{y}_i (f_i(\bar{x}) - \epsilon) + K\mu\frac{\epsilon}{2} \\
&= f_0(\bar{x}) + K\mu\frac{\epsilon}{2} \\
&\geq f_0(\bar{x}) + \mu \sum_{i=1}^{K} \frac{f_i(\bar{x})^2}{2\epsilon}
\end{aligned}
$$

which leads to a contradiction. This completes the proof. ∎

Therefore although ϵ-feasibility is not guaranteed from the statement of the above proposition, the degree of infeasibility will be within ϵ for at least one constraint. The question we address next is whether we can provide a bound on the degree of infeasibility of ϵ-violated constraint. The answer is affirmative and moreover the bound can be characterized as a linear function of ϵ. To state this result in a convenient form we need three intermediate results that we state next.

Lemma 8. Let (x^δ, y^δ) be a primal-dual optimal pair for a δ perturbation of the original problem where $\frac{\epsilon}{2} \leq \delta$. Then, for any $x \in \Re^n$

$$\tilde{F}(x, \mu, \epsilon) \geq f_0(x^\delta) + \sum_{i=1}^{K} (\mu - y_i^\delta)(f_i^+(x) - \delta) \tag{2.35}$$

where $f_i^+(x) = \max\{0, f_i(x)\}$.

Proof. We proceed as in Proposition 4. Since (x^δ, y^δ) is a primal-dual optimal pair for a δ-perturbation of [NLP], using the first-order optimality conditions we have:

$$\nabla f_0(x^\delta) = -\sum_{i=1}^{K} y_i^\delta \nabla f_i(x^\delta) \tag{2.36}$$

$$y_i^\delta (f_i(x^\delta) - \delta) = 0 \quad \forall \ i = 1, \ldots, K \tag{2.37}$$

$$y_i^\delta \geq 0 \quad \forall \ i = 1, \ldots, K \tag{2.38}$$

$$f_i(x^\delta) \leq \delta \quad \forall \ i = 1, \ldots, K \tag{2.39}$$

Using (2.20)-(2.36)-(2.37) and the definition of $\tilde{F}$, we obtain

$$\begin{aligned}
\tilde{F}(x,\mu,\epsilon) &\geq f_0(x^\delta) + \nabla f_0(x^\delta)^T(x-x^\delta) + \sum_{i=1}^{K} \tilde{p}(f_i^+(x)) \\
&= f_0(x^\delta) - \sum_{i=1}^{K} y_i^\delta \nabla f_i(x^\delta)^T(x-x^\delta) + \sum_{i=1}^{K} \tilde{p}(f_i^+(x)) \quad \text{(using (2.36))} \\
&\geq f_0(x^\delta) - \sum_{i=1}^{K} y_i^\delta (f_i(x) - f_i(x^\delta)) + \sum_{i=1}^{K} \tilde{p}(f_i^+(x)) \quad \text{(using (2.20))} \\
&= f_0(x^\delta) - \sum_{i=1}^{K} y_i^\delta (f_i(x) - \delta) + \sum_{i=1}^{K} \tilde{p}(f_i^+(x)) \quad \text{(using (2.37))}
\end{aligned}$$

But by Lemma 1 $\tilde{p}(f_i^+(x)) \geq \mu(f_i^+(x) - \delta)$, then

$$\tilde{F}(x,\mu,\epsilon) \geq f_0(x^\delta) - \sum_{i=1}^{K} y_i^\delta (f_i(x) - \delta) + \mu \sum_{i=1}^{K} (f_i^+(x) - \delta)$$

But since $f_i(x) \leq f_i^+(x)$ we have

$$\tilde{F}(x,\mu,\epsilon) \geq f_0(x^\delta) - \sum_{i=1}^{K} y_i^\delta (f_i^+(x) - \delta) + \mu \sum_{i=1}^{K} (f_i^+(x) - \delta)$$

Hence the result is established. ∎

Lemma 9. Let $\bar{x}$ be an optimal solution to

$$\min_{x \in X} \tilde{F}(x,\mu,\epsilon)$$

for some μ and ϵ. Then

$$\tilde{F}(\bar{x},\mu,\epsilon) \leq f_0(x^\delta) + K\mu\frac{\epsilon}{2} \tag{2.40}$$

for $\frac{\epsilon}{2} \leq \delta \leq \epsilon$.

Proof. Simply observe that by optimality of $\bar{x}$ we have

$$\tilde{F}(\bar{x},\mu,\epsilon) \leq \tilde{F}(x,\mu,\epsilon) \quad \forall \quad x \in X$$

In particular,

$$\tilde{F}(\bar{x},\mu,\epsilon) \leq \tilde{F}(x^\delta,\mu,\epsilon)$$

Then from the assumed ϵ-feasibility of x^δ, the result follows. ∎

Lemma 10. Let $\bar{x}$ be a solution to

$$\min_{x \in X} \tilde{F}(x, \mu, \epsilon)$$

for some μ and ϵ. Then

$$\sum_{i=1}^{K} (\mu - y_i^\delta)(f_i^+(\bar{x}) - \delta) \leq K\mu\frac{\epsilon}{2} \tag{2.41}$$

for $\frac{\epsilon}{2} \leq \delta \leq \epsilon$.

Proof. Follows directly from Lemmata 8 and 9. ∎

Proposition 11. Let $\bar{x}$ be a solution to

$$\min_{x \in X} \tilde{F}(x, \mu, \epsilon)$$

for some μ and ϵ. Then for $\mu > \max_i y_i^\epsilon$ and large enough,

$$f_i(\bar{x}) \leq \beta_i \epsilon \tag{2.42}$$

where the constant β_i is given as

$$\beta_i = 1 + \frac{K\mu/2 + \sum_{k \in K_{-i}} (\mu - y_k^\epsilon)}{\mu - y_k^\epsilon} \quad \forall \; i = 1, \ldots, K \tag{2.43}$$

and $K_{-i} = \{1, \ldots, i-1, i+1, \ldots, K\}$.

Proof. Observe that $f_i^+(\bar{x}) - \delta \geq -\epsilon$ for $i = 1, \ldots, K$. Let $\delta = \epsilon$ and the result can be obtained using (2.41) and straightforward algebraic manipulation. ∎

Therefore, we showed using simple arguments that the degree of infeasibility in any ϵ-violated constraint can be bounded above as a function of ϵ provided that the penalty parameter μ is larger than the maximum of Lagrange multipliers to the perturbed problem. This also implies that the error in the complementary slackness condition is also bounded by a term linear in ϵ. Truemper [1975] showed that it is possible to attain ϵ-feasibility if the penalty parameter μ is chosen as a multiple of the threshold μ^* where

$\mu^* > \max_i y_i^*$ with the classical quadratic penalty function. We now give a similar result using the linear-quadratic penalty function. This result is stronger than Proposition 11 and establishes the main approximate exactness property of the linear-quadratic penalty function.

Proposition 12. Let $\tilde{x}$ be an optimal solution to the smooth penalty problem [SEP] for some μ and ϵ. Then $\tilde{x}$ will be ϵ-feasible if μ is chosen such that

$$\mu = \frac{\mu^*}{\kappa} \tag{2.44}$$

where κ is a constant given by

$$\kappa = \frac{1 - K^{\frac{1}{2}}}{1 - K} \tag{2.45}$$

Proof. We first define the index sets

$$\mathcal{A} = \{i | f_i(\tilde{x}) \leq \epsilon\},$$

the set of indices corresponding to constraints which are satisfied or violated to within ϵ at $\tilde{x}$ and

$$\mathcal{V} = \{i | f_i(\tilde{x}) > \epsilon\},$$

the set of indices corresponding to constraints violated beyond ϵ at $\tilde{x}$. We will assume that the cardinality of the set $\mathcal{V}$ is at least one and argue by contradiction. Having defined the two index sets, we can write the objective function of the smooth penalty problem [SEP] at the assumed minimum $\tilde{x}$ as

$$f_0(\tilde{x}) + \mu \sum_{i \in \mathcal{V}} (f_i(\tilde{x}) - \frac{\epsilon}{2}) + \mu \sum_{i \in \mathcal{A}} \frac{f_i(\tilde{x})^2}{2\epsilon}$$

We consider the linear term first. The linear term can be rewritten as

$$\mu \sum_{i \in \mathcal{V}} (f_i^+(\tilde{x}) - \frac{\epsilon}{2}) = \mu^* \sum_{i \in \mathcal{V}} f_i^+(\tilde{x}) + (\mu \sum_{i \in \mathcal{V}} (f_i^+(\tilde{x}) - \frac{\epsilon}{2}) - \mu^* \sum_{i \in \mathcal{V}} f_i^+(\tilde{x}))$$

Considering the term in parentheses and recalling the definition of μ as a a function of μ^* we get

$$(\mu \sum_{i \in \mathcal{V}} (f_i^+(\tilde{x}) - \frac{\epsilon}{2}) - \mu^* \sum_{i \in \mathcal{V}} f_i^+(\tilde{x})) = \mu^* (\sum_{i \in \mathcal{V}} \frac{(f_i^+(\tilde{x}) - \frac{\epsilon}{2})}{\kappa} - f_i^+(\tilde{x}))$$

Now we define $t = f_i^+(\tilde{x})$, the term in the summation becomes

$$\frac{t - \frac{\epsilon}{2}}{\kappa} - t$$

Since $f_i(\tilde{x}) > \epsilon$ for all $i \in \mathcal{V}$,

$$\frac{t - \frac{\epsilon}{2}}{\kappa} - t > \frac{\epsilon}{2\kappa} - \epsilon$$

By definition $\kappa < \frac{1}{2}$ which makes the right-hand side of the above inequality positive. Since the cardinality of the set $\mathcal{V}$ is at least one we have

$$\mu(\sum_{i\in\mathcal{V}}(f_i^+(\tilde{x}) - \frac{\epsilon}{2}) > \mu^* \sum_{i\in\mathcal{V}} f_i^+(\tilde{x}) + \mu^*(\frac{\epsilon}{2\kappa} - \epsilon) \tag{2.46}$$

Now we consider the quadratic term. The quadratic term can also be rewritten as

$$\mu \sum_{i\in\mathcal{A}} \frac{f_i(\tilde{x})^2}{2\epsilon} = \mu^* \sum_{i\in\mathcal{A}} f_i^+(\tilde{x}) + (\mu \sum_{i\in\mathcal{A}} \frac{f_i^+(\tilde{x})^2}{2\epsilon} - \mu^* \sum_{i\in\mathcal{A}} f_i^+(\tilde{x}))$$

Again considering the term in parentheses,

$$(\mu \sum_{i\in\mathcal{A}} \frac{f_i^+(\tilde{x})^2}{2\epsilon} - \mu^* \sum_{i\in\mathcal{A}} f_i^+(\tilde{x})) = \mu^*(\sum_{i\in\mathcal{A}} \frac{f_i^+(\tilde{x})^2}{2\epsilon\kappa} - f_i^+(\tilde{x}))$$

Defining $t = f_i^+(\tilde{x})$, we get the term

$$\frac{t^2}{2\epsilon\kappa} - t$$

which is a convex function in t. The minimum is attained at $t^* = \epsilon\kappa$ — recall that $\kappa < \frac{1}{2}$ — at which point the function takes the value $-\frac{\epsilon\kappa}{2}$. Since this lower bound is negative following the positivity of ϵ and κ and the cardinality of the set $\mathcal{A}$ is assumed less than $K - 1$, the following inequality can be written

$$\mu \sum_{i\in\mathcal{A}} \frac{f_i(\tilde{x})^2}{2\epsilon} \geq \mu^* \sum_{i\in\mathcal{A}} f_i^+(\tilde{x}) - (K-1)\mu^*(\frac{\epsilon\kappa}{2}) \tag{2.47}$$

Now combining inequalities (2.46) and (2.47) we get

$$\begin{aligned} & f_0(\tilde{x}) + \mu \sum_{i\in\mathcal{V}}(f_i(\tilde{x}) - \frac{\epsilon}{2}) + \mu \sum_{i\in\mathcal{A}} \frac{f_i(\tilde{x})^2}{2\epsilon} \\ > \quad & f_0(\tilde{x}) + \mu^* \sum_{i=1}^{K} f_i^+(\tilde{x}) + \mu^*(\frac{\epsilon}{2\kappa} - \epsilon) - (K-1)\mu^*(\frac{\epsilon\kappa}{2}) \end{aligned}$$

$$
\begin{aligned}
&= f_0(\tilde{x}) + \mu^* \sum_{i=1}^{K} f_i^+(\tilde{x}) \\
&> f_0(\tilde{x}) + \sum_{i=1}^{K} y_i^* f_i(\tilde{x}) \\
&\geq f_0(x^*) + \sum_{i=1}^{K} y_i^* f_i(x^*) \\
&= f_0(x^*) \\
&= f_0(x^*) + \sum_{i=1}^{K} \tilde{p}(x^*)
\end{aligned}
$$

The first equality follows since

$$(\frac{\epsilon}{2\kappa} - \epsilon) - (K-1)(\frac{\epsilon\kappa}{2}) = 0$$

by definition of κ. The second inequality follows from the definition of μ^*. The third inequality along with the second equality follow from the definition of Lagrange multipliers. The last equality follows from the feasibility of x^*, thus giving a contradiction. Hence the set $\mathcal{V}$ must be empty. ∎

We remark that the constant κ in Truemper's result involves the parameter ϵ whereas our constant does not. This is due to our smoothing device which already incorporates ϵ. Therefore, for a given problem, the threshold value of μ to attain ϵ-feasibility is a constant independent of ϵ. The existence of a threshold value for the penalty parameter μ indicates that an ϵ-feasible solution can be computed in a single or at most a finite number of minimizations. An interesting consequence of Proposition 12 can be captured in the following result.

Proposition 13 Let (x^*, y^*) be a primal-dual optimal pair for [NLP] and and μ^* be such that $\mu^* > y_i^*$ for all $i = 1, \ldots, K$. Furthermore let $\tilde{x}$ be a solution to [SEP]. Then for $\mu = \frac{\mu^*}{\kappa}$ where κ is given by (2.45),

$$0 \leq f_0(x^*) - f_0(\tilde{x}) \leq K\mu\epsilon. \tag{2.48}$$

Proof. Since $\kappa < 1$, $\mu > \mu^*$. Hence, solving [EP] using $\mu = \frac{\mu^*}{\kappa}$ produces an optimal solution to the original problem. The rest of the proof follows from Proposition 3. ∎

Another interesting result can be stated if in addition to assumptions **A1** and **A2**, the following assumption is made.

A3. The problem [NLP] is a polyhedral convex program, Rockafellar [1970], and has a unique Lagrange multiplier.

The convex conjugate $\tilde{p}^*$ of $\tilde{p}$ is crucial to the forthcoming analysis and is given by

$$\tilde{p}_i^*(y_i) = \begin{cases} \frac{\epsilon}{2\mu} y_i^2 & \text{for } 0 \leq y_i \leq \mu \\ \infty & \text{otherwise} \end{cases} \tag{2.49}$$

Proposition 14. Under assumption **A3** let y^* be the unique Lagrange multiplier of [NLP]. Let $\tilde{x}$ be a solution of [SEP] for some ϵ. Then y^* can be computed as

$$y_i^* = \frac{dp_\epsilon}{dt}(\tilde{u}_i) = \max[0, \frac{\mu f_i(\tilde{x})}{\epsilon}] \qquad \forall \quad i = 1, \ldots, K \tag{2.50}$$

provided that

$$\mu \geq y_i^* \qquad \forall \quad i = 1, \ldots, K$$

and

$$\epsilon \leq \epsilon^*.$$

where ϵ^* is a positive scalar.

Proof. The proof is very similar to the proof of Proposition 2 of Bertsekas [1975]. Using Fenchel Duality Theorem [85, Th. 31.1] We have the following

$$\inf_{x \in X}\{f_0(x) + \sum_{i=1}^{K} p_{\epsilon_i}(f_i(x))\} = \inf_u\{q(u) + \sum_{i=1}^{K} p_{\epsilon_i}(u_i)\} = \max_y\{g(y) - \sum_{i=1}^{K} p_{\epsilon_i}^*(y_i)\} \tag{2.51}$$

where

$$q(u) = \inf_{x \in X}\{f_0(x) | f_i(x) \leq u\}$$

is the *perturbation function* corresponding to problem [NLP] and g is the *dual functional* defined as

$$g(y) = \inf_u\{q(u) + \sum_i u_i y_i\}$$

where $y \in \Re^K$. By Proposition 2 of Bertsekas [1975], to obtain a Lagrange multiplier $y^* \in \Re^K$ of the original problem [NLP] via (2.31), it is necessary and sufficient that there

exists a vector $\tilde{y} \in \Re^K$ such that

$$0 \in \partial g(\tilde{y}) \quad \text{and} \quad 0 \in \partial(g - \sum_{i=1}^{K} p^*_{\epsilon_i})(\tilde{y}) \tag{2.52}$$

But under assumption **A3** $\epsilon^* \mathcal{B} \in \partial g(y^*)$ for some $\epsilon^* > 0$ where $\mathcal{B}$ is the unit ball in $\Re^K$. Thus, by the definition of the convex conjugate p^*_ϵ, y^* is a maximizer of g and thus a Lagrange multiplier vector) and of $\partial(g - \sum_{i=1}^{K} p^*_i)(y^*)$ (i.e., y^* maximizes the right-hand side of (2.51)) provided that

$$\mu \geq y^*_i \qquad \forall \quad i = 1, \ldots, K$$

and

$$\epsilon \leq \epsilon^*.$$

Thus, the result follows. ∎

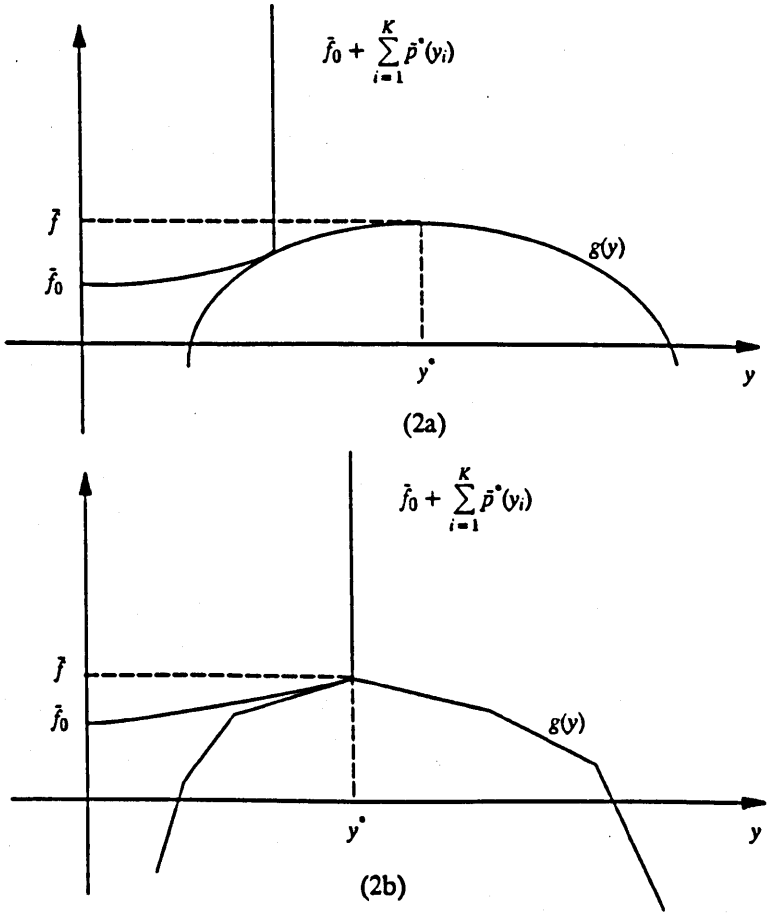
This result can hardly be considered new. A very similar result is given in Bertsekas [1975] using the quadratic penalty function. However, the main difference is that we use a linear-quadratic penalty function with favourable approximate exactness properties and we are able to recover the precise characterization of the threshold value for the penalty parameter μ.

We provide a visual aid in Figure 2.2 to understand the ideas in Proposition 14. As can be observed from the Figure 2.2a, a solution to the original problem cannot be expected by solving [SEP]. However, the situation depicted in Figure 2.2b clearly shows that the threshold value for the penalty parameter to obtain a Lagrange multiplier to the original problem by solving the smooth penalty problem can be characterized in terms of the magnitude of the Lagrange multiplier vector under the assumptions imposed for Proposition 14.

2.4 Conclusions

We considered in this chapter a smoothing approximation to the ℓ_1 exact penalty function and investigated its properties. The resulting smooth penalty function inherits some inter-

esting approximate exactness properties from its nondifferentiable counterpart. The idea of ϵ-feasibility suggests an iterative scheme where we start with a relatively large value of ϵ and a suitable value of μ and solve the resulting penalty problems until a desirable degree of infeasibility and optimality is achieved. We develop an application of this methodology in Chapter 3 and specialize it to network flow problems with side constraints.



$$\inf_{x \in X}\left\{ f_0(x) + \sum_{i=1}^{K} \tilde{p}(f_i(x)) \right\} = \max_{y}\left\{ g(y) - \sum_{i=1}^{K} \tilde{p}^*(y_i) \right\}$$

$$\tilde{f}_0 \stackrel{def}{=} \inf_{x \in X}\left\{ f_0(x) + \sum_{i=1}^{K} \tilde{p}(f_i(x)) \right\}$$

$$\tilde{f} \stackrel{def}{=} \inf_{x \in X}\{ f_0(x) \mid f_i(x) \leq 0 \forall\, i = 1, \ldots, K \}$$

Figure 2.2: Geometric interpretation

Chapter 3

A Smooth Penalty Function Algorithm for Constrained Network Flows

3.1 Introduction

In Chapter 2 we analyzed the exactness properties of the linear-quadratic penalty function as an approximation to the 1-norm exact penalty function. In this chapter we describe the design of an algorithm for solving large scale constrained network flow problems based on the linear-quadratic penalty function. We use the linear-quadratic penalty function to eliminate non-network (side) constraints from the constraint set. We solve a sequence of the resulting nonlinear network problems to get a (approximate) solution to the original problem. We remark that the penalty method developed here remains applicable to other classes of large scale mathematical programs with special substructures. A related class of problems that received increasing attention recently is the stochastic programming problem with network recourse; see Mulvey and Vladimirou [1988] or Nielsen and Zenios [1990]. It has also been revealed, due to the study of Bixby and Fourer [1988], that several of the well-known linear programming models in the literature have large embedded network substructures.

Our design revolves around two main ideas. First we use the linear-quadratic penalty

function to eliminate non–network constraints from the constraint set. Second, we use a simplicial decomposition algorithm to solve the penalty problem, and hence induce separability in the objective function. For problems with replicated network structures — like the multicommodity network flow problem or the stochastic programs with network recourse — this design also induces separability of the constraint set and can be implemented on parallel computers.

The simplicial decomposition algorithm that we use in order to induce separability was first proposed in Holloway [1974] as an extension to the linearization technique of Frank–Wolfe. Significant enhancements were added by Von Hohenbalken [1977], a memory–efficient variant was developed by Hearn, Lawphongpanich and Ventura [1987], and an inexact variant was proposed in Mulvey, Zenios and Ahlfeld [1990] where the algorithm was specialized for network structures. The use of simplicial decomposition as a device for inducing separability of large–scale problems appears to be a novelty of our approach.

This chapter is organized as follows. The application of the linear-quadratic smoothing technique to multicommodity network flow problems is developed in section 3.2. Section 3.3 provides information on important implementation aspects, like the linesearch procedure, the update of penalty parameters and the linear algebra computations. Section 3.4 presents numerical results with our implementation of the algorithm. Comparisons with alternative solution methods, like interior point algorithms, for a large-scale application are given in section 3.5. The technique is then specialized to solve network flow problems with a single commodity and side constraints in section 3.6. Numerical results are given with two sets of test problems: (1) constrained matrix estimation problems (2) NETLIB linear programming problems. Concluding remarks are given in section 3.7.

3.2 The Linear–Quadratic Penalty Algorithm for Multicommodity Flows

3.2.1 Problem Definition

We consider the following nonlinear program:

[MCNFP]

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & Ax = b \\ & Ex \leq d \\ & 0 \leq x \leq u \end{aligned}$$

where:

f: $\Re^n \rightarrow \Re$ is the objective function, assumed to be convex and continuously differentiable

$x \in \Re^n$ is the vector of decision variables which represent flows on a graph,

A is an $m \times n$ constraint matrix with network structure (i.e. it could be the node–arc incidence matrix of a network flow problem, or it could be a block–diagonal matrix where each block is a node–arc incidence matrix for a multicommodity network flow problem). We assume that $A = \text{diag}[A_1, A_2, \ldots, A_K]$, where $K \geq 1$, and each A_l is a node–arc incidence matrix.

E is the $s \times n$ matrix of side (i.e. non-network) constraints,

$l, u \in \Re^n$ are bounds on the variables,

$b \in \Re^m, d \in \Re^s$ are the right–hand side coefficients of the constraints.

Let also

$$X = \{x | Ax = b, 0 \leq x \leq u\}\text{, and,}$$

$$\Omega = \{x | x \in X, Ex \leq d\}.$$

We consider the following smoothed penalty version of the nonlinear program [MCNFP].

[PNLP]

$$\min_{x \in X} \Phi_{\mu,\epsilon}(x) = f(x) + \mu \sum_{j=1}^{s} \tilde{p}(y_j)$$

where $y_j = (Ex - d)_j$, for $j = 1, 2, \ldots, s$ and μ is a positive real number.

The use of smooth penalty function to eliminate side constraints motivates the following iterative framework:

The Linear–Quadratic Penalty Algorithm

Step 0 (Initialization.) Find an initial feasible solution for the relaxed problem

[**RMCNFP**]

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & Ax = b \\ & 0 \leq x \leq u \end{aligned}$$

Set $k = 0$, let x^0 be the optimal solution of this problem. If $x^0 \in \Omega$ stop. Otherwise choose $\mu^0 > 0$ $\epsilon^0 > 0$, go to Step 1.

Step 1 Using the violation $y_j = (Ex - d)_j$ for all j as the starting point for evaluating the penalty function solve the problem

$$\min_{x \in X} \Phi_{\mu,\epsilon}(x) = f(x) + \mu \sum_{j=1}^{s} \tilde{p}(y_j)$$

Let x^{*k} denote the optimal solution.

Step 2 If x^{*k} satisfies stopping criteria terminate. Otherwise, let $x^{k+1} \leftarrow x^{*k}$, update the penalty parameters μ and ϵ, set $k \leftarrow k + 1$ and proceed from Step 1.

Details on this general algorithmic framework are completed in the following sections.

Most of the computational effort of the algorithm is in solving the smoothed penalty problem [**PNLP**] in Step 1. This problem is once continuously differentiable, and can be solved with any general purpose nonlinear programming algorithm like truncated Newton, Dembo [1987] or a variant of truncated Newton as proposed in Toint and Tuyttens [1990]. However, the penalty function $\tilde{p}(y)$ is non-separable in x (recall that $y_j = (Ex - d)_j$). Hence, the fact that the original problem [**NLP**] is such that X is a Cartesian product set, (i.e. $A = \text{diag}[A_1, A_2, \ldots, A_K]$, where $K > 1$) gives one the incentive to use an algorithm for [**PNLP**] that is based on a linearization of the smoothed exact penalty function. In this way the algorithm decomposes for each subproblem. Examples of such algorithms are the reduced gradient, Successive Linear Programming (SLP) and simplicial decomposition. We choose to solve [**PNLP**] using simplicial decomposition. A thorough description of the simplicial decomposition algorithm can be found in the papers by Von Hohenbalken [1977], Hearn, Lawphongpanich and Ventura [1987] and Mulvey, Zenios and Ahlfeld [1990].

3.2.2 Linearization via Simplicial Decomposition

Simplicial decomposition iterates by solving a sequence of linear problems to generate vertices of the polytope X. A nonlinear master problem optimizes the penalized objective function $\Phi_{\mu,\epsilon}$ on the simplex specified by the vertices generated by the subproblems. The simplicial decomposition algorithm for [**PNLP**] at the k-th iteration of the LQP algorithm may be stated as follows:

The Simplicial Decomposition Algorithm

Step 0 Set $\nu = 0$, and use $z^0 \leftarrow x^k \in X$ as the starting point. Let $Y = \emptyset$, and $v \leftarrow 0$ denote the set of generated vertices and its cardinality, respectively.

Step 1 (Linearized subproblem.) Compute the gradient of the penalty function $\Phi_{\mu,\epsilon}$ at the current iterate z^ν and solve a linear program to get a vertex of the constraint set, i.e. solve for $y^* = \arg\min_{y \in X} y^T \nabla \Phi_{\mu,\epsilon}(z^\nu)$ and let $Y = Y \cup \{y^*\}$, $v \leftarrow v + 1$

Step 2 (Nonlinear master problem.) Using the set of vertices Y to represent a simplex over the constraint set X find an optimizer of the penalized objective function $\Phi_{\mu,\epsilon}$ over this subset of X. Let $w^* = \arg\min_{w \in W_v} \Phi_{\mu,\epsilon}(Bw)$ where $W_v = \{w_i | \sum_{i=1}^{v} w_i = 1, w_i \geq 0 \ \forall \ i = 1,2,\ldots,v\}$ and $B = [y^1|y^2|\ldots|y^v]$ is the basis for the simplex generated by the set of vertices Y. The optimizer of $\Phi_{\mu,\epsilon}$ over the simplex is given by $z^{\nu+1} = Bw^*$.

Step 3 Let $\nu \leftarrow \nu + 1$, and return to Step 1.

At Step 1 the algorithm solves a linear approximation to the nonlinear program. If the set X has a Cartesian product structure the problem can be solved independently for each block of the constraint matrix A:

Step 1 (Decomposed linear subproblems.)

For each $\ell = 1, 2, \ldots, K$ solve

$$\begin{aligned} \text{Minimize} \quad & y_\ell^T \nabla_\ell \Phi_{\mu,\epsilon}(z^\nu) \\ \text{subject to} \quad & \\ & A_\ell y_\ell = b_\ell \\ & 0 \leq y_\ell \leq u_\ell \end{aligned}$$

We use ℓ to index the ℓ-th block of the constraint matrix $A = \text{diag}[A_1, A_2, \ldots, A_K]$. The subproblems are linear network programs for each commodity and thus may be solved in parallel.

The nonlinear master problem in Step 2 is of much smaller size than the original [NLP]. Furthermore, it has a very simple constraint structure: non-negativity constraints and a simplex equality constraint. However, this problem is likely to present difficulties due to the ill-scaled nature of the objective function as μ gets larger. Also, the objective function is only once continuously differentiable. Discussions on the efficient solution of the master problem are deferred until section 3.3.3.

3.3 Numerical Issues

There are three components of the LQP algorithm that deserve special attention in the implementation: 1. The scheme for estimating and updating the multipliers $\{\mu^k\}$ and the tolerance $\{\epsilon^k\}$, 2. The linesearch algorithm for a piecewise linear-quadratic function, and 3. The nonlinear programming algorithm for solving the unconstrained master problem. The master problem is typically of small size but may be ill-conditioned. The choices made for these three components are crucial for a robust implementation, and we discuss all of them in detail here.

3.3.1 Penalty Parameter Adjustments

Suppose $x^k \in X$, μ^k, ϵ^k are given. Let $y^k = Ex^k - d$ and define

$$\mathcal{A}(x^k) = \{j | 0 < y_j^k \leq \epsilon^k\}, \text{the set of active constraint indices}$$

$$\mathcal{V}(x^k) = \{j | y_j^k > \epsilon^k\}, \text{the set of violated constraint indices}$$

$$\mathcal{S}(x^k) = \{j | y_j^k \leq 0\}, \text{the set of satisfied constraint indices}$$

The iterate x^k is termed *ϵ–feasible* if $\mathcal{V}(x^k) = \emptyset$. Consider the first order optimality conditions corresponding to the original problem [**MCNFP**] and to the penalty problem [**PNLP**].

Associate the multipliers : λ with constraints $Ax = b$,
$\omega \geq 0$ with constraints $Ex \leq d$ and,
$\theta^u, \theta^l \geq 0$ with constraints $0 \leq x \leq u$,

and let $g = \text{grad} f$.

The first-order optimality conditions for [**MCNFP**] are then:

$$g(x) + A^T\lambda + E^T\omega + \theta^u - \theta^l = 0 \tag{3.1}$$

$$Ax = b \tag{3.2}$$

$$Ex \leq d \tag{3.3}$$

$$l \leq x \leq u \tag{3.4}$$

$$w \geq 0 \ ; \ \theta^u \geq 0 \ ; \ \theta^l \geq 0 \tag{3.5}$$

$$w(Ex - d) = 0 \tag{3.6}$$

$$\theta^u(x - u) = 0 \tag{3.7}$$

$$\theta^l x = 0 \tag{3.8}$$

If we assume strict complementarity and second-order sufficiency then $x^*, \omega^*, \lambda^*, \theta^*$ satisfying the above are unique. We compare these conditions with the optimality conditions of the ϵ-exact penalty problem [**PNLP**]. Associating λ_ϵ with constraints $Ax = b$ and $\theta_\epsilon^u, \theta_\epsilon^l \geq 0$ with constraints $0 \leq x \leq u$ in [**PNLP**], we have:

$$g(x) + \mu \sum_{j=1}^{s} \nabla \tilde{p}(y_j) + A^T\lambda_\epsilon + \theta_\epsilon^u - \theta_\epsilon^l = 0 \tag{3.9}$$

$$Ax = b \tag{3.10}$$

$$l \leq x \leq u \tag{3.11}$$

$$\theta_\epsilon^u(x-u)=0 \tag{3.12}$$

$$\theta_\epsilon^l(x-l)=0 \tag{3.13}$$

$$\theta_\epsilon^u \geq 0 \ ; \ \theta_\epsilon^l \geq 0 \tag{3.14}$$

Letting $\mathcal{A}(x)=\{j|0<y_j\leq\epsilon\}$ be the set of active constraints and $\mathcal{V}(x)=\{j|y_j>\epsilon\}$ be the set of violated constraints, the above condition (3.9) reduces to:

$$g(x_\epsilon)+\frac{\mu}{\epsilon}\sum_{j\in\mathcal{A}(x_\epsilon)}(E_jx_\epsilon-d_j)E_j^T+\mu\sum_{j\in\mathcal{V}(x_\epsilon)}E_j^T+A^T\lambda_\epsilon+\theta_\epsilon^u-\theta_\epsilon^l=0 \tag{3.15}$$

where x_ϵ denotes a solution to the optimality conditions of the penalized problem [**PNLP**]. Now let

$$(\omega_\epsilon)_j=\begin{cases}\mu & \text{for } j\in\mathcal{V}(x_\epsilon)\\ \frac{\mu(E_jx_\epsilon-d_j)}{\epsilon} & \text{for } j\in\mathcal{A}(x_\epsilon)\\ 0 & \text{otherwise}\end{cases} \tag{3.16}$$

then the above reduces to

$$g(x_\epsilon)+E^T\omega_\epsilon+A^T\lambda_\epsilon+\theta_\epsilon^u-\theta_\epsilon^l=0,$$

which is reminiscent of the Kuhn-Tucker condition (3.1) for optimality in the original problem. Thus a solution x_ϵ to the ϵ-exact penalty problem and the multipliers $\omega_\epsilon,\lambda_\epsilon,\theta_\epsilon^u,\theta_\epsilon^l$ defined above, satisfy conditions (3.1),(3.2),(3.4),(3.5),(3.6),and (3.7) of the conditions for optimality of the original problem. Optimality for the problem [**MCNFP**] would be assured if we had primal feasibility (i.e. $Ex_\epsilon\leq d$; equation (3.3)) and complementary slackness (i.e. $\omega_\epsilon(Ex_\epsilon-d)=0$; equation (3.6))

In solving [**PNLP**] we are content to achieve an x_ϵ that satisfies (3.10)-(3.14), that is ϵ-feasible, i.e. $\mathcal{V}(x_\epsilon)=\emptyset$ (and hence $Ex_\epsilon-d\leq\epsilon$) and that (almost) satisfies complementary slackness (3.6). Since, by construction, $\omega_\epsilon\geq 0$, a necessary condition for ϵ-optimality is

$$Ex_\epsilon-d\leq\epsilon \quad (\epsilon-\text{feasibility})$$

which implies (by definition of ω_ϵ)

$$\omega_\epsilon(Ex_\epsilon-d)\leq\mu\epsilon \quad (\epsilon-\text{complementary slackness}).$$

Thus, for ϵ small enough, $\mu\epsilon$ will be small. The above analysis gives us some insight as to how two of the key parameters in an algorithm for [**PNLP**] may be adjusted from

iteration to iteration. First, ϵ should start out relatively large and should be reduced so that $\mu\epsilon$ becomes an acceptable complementary slackness error in the limit. If no constraint is violated (i.e. $\mathcal{V}(x_\epsilon) = \emptyset$) then leave μ unchanged. If at least one constraint is violated increase μ according to the formula

$$\mu \leftarrow \max\{\frac{\mu(E_j x_\epsilon - d_j)}{\epsilon}\}$$

By the first–order optimality conditions of [**PNLP**] the Lagrange multiplier estimates are computed as follows:

$$\omega_j^k \equiv 0 \ \forall\, j \in \mathcal{S}(x^k) \tag{3.17}$$

$$\omega_j^k \equiv \mu \ \forall\, j \in \mathcal{V}(x^k) \tag{3.18}$$

$$\omega_j^k \equiv \mu \frac{y_j}{\epsilon^k} \ \forall\, j \in \mathcal{A}(x^k) \tag{3.19}$$

We distinguish the following two cases:

1 . If $\mathcal{V}(x^k) = \emptyset$, this is taken to be an indication that the magnitude of the penalty parameter μ was adequate in the previous iteration since ϵ–*feasibility* is achieved. In this case the infeasibility tolerance ϵ could be reduced.

2 . If $\mathcal{V}(x^k) \neq \emptyset$, ϵ–*feasibility* was not achieved and the penalty parameter μ would have to be increased proportional to the degree of infeasibility. Let $\gamma = \eta\epsilon^k$ be a target degree of infeasibility where $\eta \in (0,1]$. By using (3.18) and (3.19) we get the following update equation for μ^{k+1}:

$$\mu^{k+1}\frac{\gamma}{\epsilon^k} = \mu^k, \tag{3.20}$$

which is equivalent to

$$\mu^{k+1} = \mu^k \frac{1}{\gamma}. \tag{3.21}$$

But, equation (3.21) would lead to identical updates of the penalty parameter regardless of the degree of infeasibility. However, we can consider, instead of (3.21), the following update equation:

$$\mu^{k+1} = \mu^k \frac{y_j}{\gamma}, \tag{3.22}$$

or equivalently,

$$\mu^{k+1} = \mu^k \frac{y_j}{\eta\epsilon^k}. \tag{3.23}$$

And considering $|\mathcal{V}(x^k)| \geq 1$, we get

$$\mu^{k+1} = \frac{\mu^k}{\eta\epsilon} \max_{j \in \mathcal{V}(x^k)} y_j. \tag{3.24}$$

In summary we have the following update procedure :

Pick η_1, $\eta_2 \in (0, 1]$

If $\mathcal{V}(x^k) = \emptyset$

$\epsilon^{k+1} \leftarrow \eta_1 \epsilon^k$

else

$\mu^{k+1} = \frac{\mu^k}{\eta_2 \epsilon^k} \max_{j \in \mathcal{V}(x^k)} y_j$.

A suitable initial value for μ can be found through some preliminary experimentation. With the linear programs we used in this study, the absolute maximum of objective function coefficients proved to be a good choice for the initialization of μ. The solution to the multicommodity network flow problem obtained by ignoring the side constraints can be used to obtain an initial value for ϵ. A reasonable choice is to pick a value equal to a fraction of the maximum of the side constraint violations, i.e. in the interval $(0, \max_{j \in \mathcal{V}(x^0)} y_j)$

3.3.2 Alternative Linesearch Procedures

We consider now the problem of minimizing the piecewise linear–quadratic function along a direction of descent. This problem is solved at every iteration of the master problem in simplicial decomposition. It can be posed as follows:

$$\min_{0 \leq \alpha \leq 1} \Phi_{\mu,\epsilon}(y + \alpha p) \tag{3.25}$$

where $p = [p_1|p_2|\ldots|p_K]^T$ is a descent direction at point y. That is, i.e. $\nabla\Phi^T_{\mu,\epsilon} \cdot p < 0$. (How to compute such a descent direction is left for the next section.)

The function $\Phi_{\mu,\epsilon}$ is once continuously differentiable and convex, and as shown in Dembo and Anderson [1989] there exists an optimal solution to (3.25) at the minimum of a quadratic segment. We may apply any one–dimensional search algorithm such as a quadratic interpolation with safeguards, see, e.g. Gill, Murray and Wright [1981]. It is likely, however,

that a linesearch based on quadratic interpolation will be inefficient when the function has a large number of linear segments, and the linesearch is started far from the minimizer. We adopted a linesearch due to Dembo and Anderson [1989]. The method attempts to build a coarse approximation to the function based on local information. In general, when far from a minimum the function is viewed as piecewise linear. At each stage the function values at two distinct points bracketing the minimum are assumed to be known. The *left hand point*, α_L, has a negative gradient and the gradient at *the right hand point*, α_R, is positive. We say that the left hand point is *quadratic(linear)* if it lies on a quadratic(linear) segment. By the *left hand line* we mean the line corresponding to the linear segment when the left point is linear. Similarly, *the right hand quadratic* corresponds to the quadratic approximation at a right hand point that is quadratic. The linesearch algorithm has the following general form:

START with $[\alpha_L, \alpha_R]$ bracketing the minimum.

COMPUTE a new approximation to the minimum $\hat{\alpha}$;

WHILE $\nabla_\alpha \Phi(y + \alpha p) \neq 0$

If $\nabla_\alpha \Phi(y + \alpha p) > 0$, $\alpha_R \longleftarrow \hat{\alpha}$

If $\nabla_\alpha \Phi(y + \alpha p) < 0$, $\alpha_L \longleftarrow \hat{\alpha}$.

Let us now examine the three cases that may arise:

CASE I *The left and right hand points are both linear;*

The function is approximated as a piecewise linear function and the next approximation is taken to be the intersection of the left and right hand lines.

CASE II *The left hand point is linear and the right hand point is quadratic(or vice versa);*

We know that the current left and right hand points are not in the neighborhood of a minimum. We therefore continue to treat the problem as if it were piecewise linear and our next approximation point is the intersection of the linear approximations at the two points.

CASE III *The left and right hand points are quadratic;*

Here we have reason to suspect that we have bracketed the quadratic segment on

which the minimum lies. We therefore fit a quadratic through the two points which will be exact in a neighborhood of the solution.

3.3.3 Computing Directions of Descent

We return now to the solution of the master problem in simplicial decomposition. It is a nonlinear problem with simple bounds and a single equality (simplex) constraint. There are several standard methods that can be used for its solution, like, for example, Bertsekas's projected Newton method [1982]. If the simplicial decomposition algorithm drops vertices that carry zero weight at the optimal solution of the master problem, then subsequent master programs are locally unconstrained. Hence, methods of unconstrained optimization can be used to compute a descent direction. A simple ratio test determines the maximum feasible step length that will not violate the bounds. This problem is studied in Mulvey, Zenios and Ahlfeld [1990]. Here we consider additional difficulties that arise due to the piecewise linear–quadratic nature of the objective function.

The master program can be written in the form:

$$\min_{w \geq 0} \Phi_{\mu,\epsilon}(Dw) \tag{3.26}$$

where $D = [y_1 - y_v | y_2 - y_v | \ldots | y_{v-1} - y_v]$ is the derived linear basis for the simplex generated by the vertices $y_1, y_2, \ldots, y_v$. We denote by w the vector $[w_1, w_2, \ldots, w_{v-1}]$ and the solution for w_v is computed as

$$w_v = 1 - \sum_{i=1}^{v-1} w_i, \tag{3.27}$$

Recall that at the current iteration we have $v-1$ active vertices (i.e. $w_i > 0$, for $i = 1, \ldots, v-1$) and the last vertex y_v lies along a direction of descent. Hence, given a point $z^\nu \in X_0$ a descent direction to (3.26) can be obtained as the solution to

$$(D^T M D)p = -D^T \nabla \Phi_{\mu,\epsilon}(z^\nu), \tag{3.28}$$

where the choice of the matrix M is discussed next.

Reduced Gradient Steps

If we choose M to be a conformable identity matrix we have

$$(D^T D)p_I = -D^T \nabla \Phi_{\mu,\epsilon}(z^\nu) \tag{3.29}$$

and p_I is the reduced gradient direction. An exact method can be used to solve for p_I. Note, however, that D is of dimension $n \times (v-1)$. In general n is very large (more than 100,000 for some of the larger test problems), whereas v starts from 1 and rarely exceeds 100. Hence $D^T D$ is a relatively small matrix of dimensions $((v-1) \times (v-1))$ which can be easily inverted once it has been explicitly formed. To avoid the computation of the product $D^T D$ we use a QR factorization of D and hence solve

$$(R^T R)p_I = -D^T \nabla \Phi_{\mu,\epsilon}(z), \tag{3.30}$$

where R is an upper triangular matrix. Numerical results with this method, and comparisons with a LU decomposition of $D^T D$ and a conjugate gradient solver are discussed in section 3.4.2.

Truncated Newton Steps

If we choose M to be the Hessian matrix ($H = \nabla^2 \Phi_{\mu,\epsilon}(z)$) we have

$$(D^T H D)p_{TN} = -D^T \nabla \Phi_{\mu,\epsilon}(z). \tag{3.31}$$

This system is not always positive definite which precludes the possibility of using a direct method(the Hessian matrix could have zero elements along the diagonal when the penalty function is at a linear segment). However, it can be solved inexactly using a conjugate gradient solver to compute a truncated Newton step.

Quasi–Newton Steps

If we choose M to be the Hessian matrix ($H_\nu = \nabla^2 \Phi_{\mu,\epsilon}(z^\nu)$) evaluated at the solution to the master problem of the previous iteration, we have

$$(D^T H_\nu D)p_{QN} = -D^T \nabla \Phi_{\mu,\epsilon}(z^\nu) \tag{3.32}$$

This approach, first suggested in Mulvey, Zenios and Ahlfeld [1990], is particularly attractive since it removes the requirement to form the product $D^T H D$ at every step of the master problem algorithm. Instead, we work with a fixed matrix $D^T H_\nu D$ which is computed only once. Unfortunately the Hessian is not guaranteed to be positive definite and the system is solved inexactly using a conjugate gradient solver.

3.3.4 Termination Criteria

In this section we examine the termination criteria used in the LQP algorithm. We show that it is possible to obtain lower and upper bounds on the optimal objective value of the orginal problem [**MCNFP**] even in the presence of inexact minimizations of the penalized objective function.

Let x^* be an optimal solution of [**MCNFP**] and $\hat{x}_{\mu,\epsilon}$ an optimal solution of [**PNLP**] for given penalty parameters μ and ϵ. Then it is well-known that, see for instance Fiacco and McCormick [1968],

$$\Phi_{\mu,\epsilon}(\hat{x}_{\mu,\epsilon}) \leq f(x^*). \tag{3.33}$$

Therefore the optimal solution of [**PNLP**] is a lower bound for the optimal objective value. But in presence of inexact minimizations of the penalized objective function $\Phi_{\mu,\epsilon}$, this is not always guaranteed to be a lower bound. Hence, we consider the first order Taylor series expansion of $\Phi_{\mu,\epsilon}$ around $z \in X$

$$\Phi_{\mu,\epsilon}(x) = \Phi_{\mu,\epsilon}(z) + (x-z)^T \nabla \Phi_{\mu,\epsilon}(z) + o(\|z\|^2) \;\; \forall x \in X \tag{3.34}$$

and ignoring the second order term define the function, $h_{\mu,\epsilon} : \Re^n \longmapsto \Re$,

$$h_{\mu,\epsilon}(x) = \Phi_{\mu,\epsilon}(z) + (x-z)^T \nabla \Phi_{\mu,\epsilon}(z) \tag{3.35}$$

Since $h_{\mu,\epsilon}$ is majorized by $\Phi_{\mu,\epsilon}$ by convexity of $\Phi_{\mu,\epsilon}$, $\min_{x \in X} h(x) \leq \Phi_{\mu,\epsilon}(\hat{x}_{\mu,\epsilon})$. Hence, by (3.33), $\min_{x \in X} h_{\mu,\epsilon}(x) \leq f(x^*)$. Hence, in the presence of inexact minimizations, a lower bound can be obtained by minimizing the linear approximation. This bound is already needed by the simplicial decomposition algorithm that generates extreme points of X by minimizing a linearized approximation to the objective function over X. Our computational experience shows that this lower bound may not be very tight. However, it is possible to obtain tighter lower bounds to the optimal value as follows. Let x^k be an arbitrary iterate and denote by $Q(\epsilon, \rho^k) = \sum_{j=1}^{s} \tilde{p}(\rho_j^k)$ where $\rho^k = E\mathrm{x}^k - d$. Recall the subproblem objective function of the simplicial decomposition algorithm:

$$\nabla \Phi_{\mu,\epsilon}(\mathrm{x}^k)^T \cdot \mathrm{x} = (\nabla f(\mathrm{x}^k) + \mu \nabla_\rho Q(\epsilon, \rho^k)^T E) \cdot \mathrm{x} \tag{3.36}$$

We define the following lower bound function

$$V(\hat{u}) = (\nabla f(\mathrm{x}^k)^T - \hat{u}E)\mathrm{x}' + \hat{u}d \tag{3.37}$$

where the vector $\hat{u}$ is given by

$$\hat{u} = -\mu \nabla_{\rho} Q(\epsilon, \rho^k) \tag{3.38}$$

and x' is the solution to the linearized subproblem, i.e., $\mathrm{x}' = \arg\min_{\mathrm{x} \in X} x^T \nabla \Phi_{\mu,\epsilon}(\mathrm{x}^k)$. Next we show that $V(.)$ provides a lower bound superior to the linearized subproblem bound. The analysis is a generalization of the result given in Brown et al. [1989]. To proceed we need the following intermediate result.

Lemma 1. Let $g : \Re^n \longmapsto \Re$ be a convex and at least once continuously differentiable function with the property that

$$g(\mathbf{0}) = 0 \tag{3.39}$$

where $\mathbf{0}$ is the zero vector. Then

$$g(y) - y^T \nabla g(y) \leq 0 \qquad \forall \ \ y. \tag{3.40}$$

Proof. Consider a first-order Taylor series expansion of g. By convexity

$$g(x) \geq g(y) + (x - y)^T \nabla g(y) \qquad \forall \ \ x, y \in \Re^n$$

In particular,

$$g(\mathbf{0}) \geq g(y) - y^T \nabla g(y)$$

However, by hypothesis $g(\mathbf{0}) = 0$, and the result follows. ∎

The assumption in Lemma on f holds for example for linear objective functions and quadratic objective functions of the form $\sum_j a_j x_j^2$.

Proposition 2. Let f be as in Lemma 1 and $\mathrm{x}' = \arg\min_{\mathrm{x} \in X} x^T \nabla \Phi_{\mu,\epsilon}(\hat{\mathrm{x}})$ where $\hat{\mathrm{x}}$ is the current iterate. Then

$$h_{\mu,\epsilon}(\mathrm{x}') \leq V(\hat{u})$$

where $h_{\mu,\epsilon}$ is given by (3.35).

Proof. We will show that

$$h_{\mu,\epsilon}(\mathrm{x}') - V(\hat{u}) \leq 0$$

Equivalently we want to show

$$\Phi_{\mu,\epsilon}(\hat{x}) + (x' - \hat{x})^T \nabla \Phi_{\mu,\epsilon}(\hat{x}) - (\nabla f(\hat{x})^T - \hat{u}E)x' - \hat{u}d \leq 0$$

Consider the left hand side. Let $\hat{\rho} = E\hat{x} - d$. By algebraic manipulation and using the definition of $\hat{u}$,

$$\begin{aligned}
&\Phi_{\mu,\epsilon}(\hat{x}) + (x' - \hat{x})^T \nabla \Phi_{\mu,\epsilon}(\hat{x}) - (\nabla f(\hat{x})^T - \hat{u}E)x' - \hat{u}d \\
&= f(\hat{x}) + \mu Q(\epsilon, \hat{\rho}) - \nabla f(\hat{x})^T \hat{x} + \hat{u}(E\hat{x} - d) \\
&= f(\hat{x}) - \nabla f(\hat{x})^T \hat{x} + \mu Q(\epsilon, \hat{\rho}) - \mu \nabla_\rho Q(\epsilon, \hat{\rho})^T (E\hat{x} - d) \\
&= \underbrace{f(\hat{x}) - \nabla f(\hat{x})^T \hat{x}}_{1} + \underbrace{\mu(Q(\epsilon, \hat{\rho}) - \nabla_\rho Q(\epsilon, \hat{\rho})^T \hat{\rho})}_{2}
\end{aligned}$$

By the assumption imposed on f, the first term is nonpositive following Lemma 1. The nonpositivity of the second term follows from the fact that $\mu > 0$ and

$$Q(\epsilon, 0) = 0$$

and by invoking Lemma 1. Hence the claim is established. ∎

We now proceed to describe the procedure for generating upper bounds in the linear-quadratic penalty algorithm. For a more general discussion on bounding exterior penalty function algorithms see Fiacco and McCormick [1968]. Define the set $R^0 = \{x \in X | Ex < d\}$ and assume that R^0 is non–empty. Let $x \in R^0$ and $\hat{x}_{\mu,\epsilon}$ be an — perhaps approximate — optimal solution of [PNLP] then a new interior point $\underline{x}$ is generated as follows: let $y = E\hat{x}_{\mu,\epsilon} - d$, and $I = \{i | y_i > 0\}$. Let $y' = Ex - d$ and $y'_i < 0 \; \forall \; i = 1, \ldots, s$. Calculate

$$\beta = \max_{i \in I} \frac{y_i}{y_i - y'_i} \tag{3.41}$$

and define

$$\underline{x} = (1 - \beta)\hat{x}_{\mu,\epsilon} + \beta x \tag{3.42}$$

It can be shown that $\underline{x} \in \Omega$ and provides an upper bound, Fiacco and McCormick [1968, Theorem 29, p. 107]. The same result also proves that the upper bound converges monotonically to the optimal objective value. Obviously this procedure requires an interior point to be generated at the beginning of the algorithm. We adopted a procedure given in Armstrong, Qi and Zenios [1990] to generate a starting interior feasible solution for multicommodity network flow problems.

We have therefore both upper and lower bounds for the optimal objective value during the execution of the algorithm. A new upper bound is computed after every penalty minimization, i.e. execution of Step 1 of the LQP algorithm. The lower bound is updated after every subproblem phase if the new lower bound is superior to the current lower bound. The algorithm terminates when both of the following error measures are within acceptable tolerance.

1. Absolute error in side constraint feasibility

$$\|Ex - d\|_\infty \leq e_{inf}$$

2. Bound gap

$$\frac{f(\underline{x}) - h_{\mu,\epsilon}(x)}{h_{\mu,\epsilon}(x)} \leq e_{gap}$$

where x is the current iterate and $\underline{x}$ is obtained from (3.42).

The ability to compute improving upper bounds is an important feature of our approach since computation can be stopped as soon as a reasonable improvement in the upper bound is achieved.

3.4 Computational Results

The LQP algorithm was implemented for the multicommodity network flow structures. We used the simplicial decomposition implementation from the system GENOS of Mulvey and Zenios [1987] to solve the penalty subproblems. A main program around simplicial decomposition sets up the penalty function; the master problem solver in GENOS has been modified to handle the factorization procedures outlined in section 3.3.3. The code – which we call GENOS/MC for Generalized Network Optimization System on Multi-Commodity networks – is written in FORTRAN 77.

In this section we report summary computational results. The intention is not to present an exhaustive list of experiments, but to highlight key aspects of the algorithm and illustrate its suitability in solving very large problems. We also provide comparisons with alternative solution algorithms for identical test problems, thus illustrating the effectiveness of the algorithm and the efficiency of our implementation.

In all computational tests, we have $e_{inf} = 10^{-5}$, and $e_{gap} = 10^{-2}$ on termination. As we remarked earlier the method does not generate good lower bounds. This explains the high value of the error e_{gap}. Solving the penalized problem [PNLP] to optimality would remedy this situation at the expense of higher computation times. We point out, however, that the answers we obtained match optimal values given by MINOS to five digits for the problems we solved with both systems.

All results reported in subsequent sections refer to CPU seconds on a VAX 6400 running VMS 5.3, unless otherwise indicated. The program was compiled with default optimization level. All reported times exclude input/output of data.

3.4.1 Test Problems

We consider two classes of test problems. One class of problems was obtained from a Military Airlift Command application. They are referred to as the Patient Distribution System (PDS) problems and are used to make decisions on the evacuation of patients from Europe. PDSt denotes a problem that models a scenario of t days. They are linear multicommodity network problems with eleven commodities. The size and complexity of the models increase with t.

The KENx problems are randomly generated multicommodity networks. They were generated by the code MNETGN of Ali and Kennington [1977]. The size and characteristics of all the test problems we solved from both classes are summarized in Table 3.1.

Test problem	Network Formulation			Linear Programming	
	Arcs	Nodes	Commodities	Rows	Columns
KEN7	73	49	49	2426	3602
KEN11	176	121	121	14694	21349
KEN13	225	169	169	28632	42659
PDS1	339	126	11	1473	3816
PDS3	1117	390	11	4593	12590
PDS5	2149	686	11	7546	23639
PDS10	4433	1399	11	15389	48763
PDS15	7203	2125	11	23375	79233
PDS20	10116	2447	11	31427	105728

Table 3.1: Test Problem Characteristics.

3.4.2 Solving the Master Problem

We experimented with the three method of section 3.3.3 for computing descent directions: Reduced gradient, Newton, quasi–Newton. We also used exact solvers(LU or QR factorizations) and an iterative solver(conjugate gradient) for computing truncated Newton directions. The performance of each combination is illustrated with two test problems PDS1 and PDS3 in Tables 3.2 and 3.3. Computing Quasi–Newton steps using a conjugate gradient solver has a distinct advantage over all other methods. It is the default master program solver of GENOS/MC.

	Linear Algebra Solver		
Step selection	**LU**	**QR**	**Conj. grad.**
Reduced gradient	319.41	318.0	244.49
Newton	NA	NA	188.14
Quasi–Newton	NA	NA	130.4

Table 3.2: Performance of various step selection strategies with PDS1.

	Linear Algebra Solver		
Step selection	**LU**	**QR**	**Conj. grad.**
Reduced gradient	3910	3924	5644
Newton	NA	NA	1998.7
Quasi–Newton	NA	NA	1239

Table 3.3: Performance of various step selection strategies with PDS3.

3.4.3 Performance of Linesearch Procedures

The nonlinear master program can use either the piecewise linear–quadratic linesearch of section 3.3.2, or a standard quadratic interpolation linesearch with safeguards that is already in GENOS. Both linesearch routines were tested and compared on PDS1. Figure 3.1 compares the two linesearch routines with respect to number of function evaluations and CPU time. Additional details are given in Appendix A. Major iterations refer to executions of Step 2 of the LQP algorithm whereas simplicial iterations is the total number of simplicial decomposition steps required for the solution of penalty problems. This comparison was done using reduced gradient steps in the solution of the master problem. First we observe from the results that the overall performance of the LQP algorithm does not depend on the choice of the linesearch procedure. This was, of course, anticipated since both linesearch procedures solve each master problem to the same level of accuracy. Second, we observe from the results of Figure 3.1 that the two methods are at par with each other with a slight advantage of the GENOS linesearch. This result is somewhat surprising since we expected the piecewise linear–quadratic linesearch to take advantage of the special structure of the penalty function. For problems with a large number of linear segments and a small number of quadratic segments this will indeed be the case. Both linesearch routines are part of GENOS/MC and a user controlled parameter determines which one is used.

3.4.4 Restart Procedures for the Network Subproblems

One of the major components of the solution time comes from the effort expended in solving the subproblems in the simplicial decomposition algorithm. The subproblems are linear network flow problems for each commodity and can be solved with the network special-

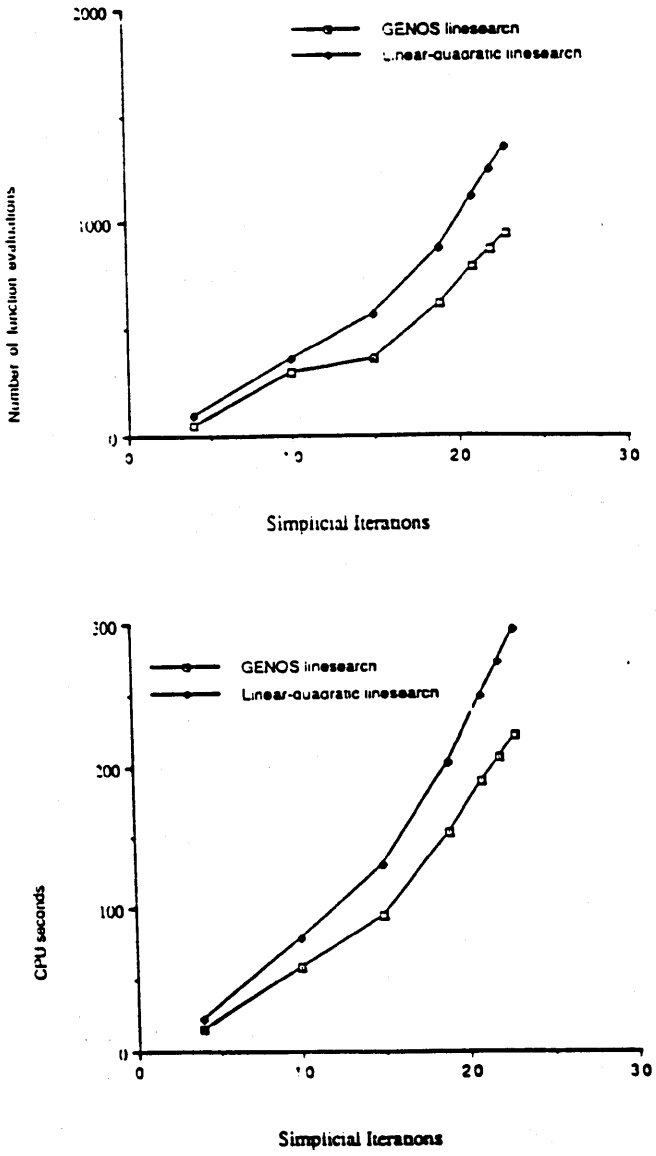


Figure 3.1: Comparison of 1.The piecewise linear/quadratic linesearch with 2.The quadratic interpolation with safeguards.

ization of the simplex method starting with an artificial basis at each subproblem phase. Alternatively, an advanced basis start procedure can be incorporated into the subproblem phase to reduce the number of pivots to optimality. This can be achieved, for instance, using the maximal basis procedure of Dembo and Klincewicz [1985]. This procedure uses a greedy heuristic to construct a basis tree from a given feasible iterate. It is also possible to save the basis arrays of the previous iteration and use the previous basis as the initial basis in the current subproblem phase. Both these procedures were implemented and tested. We illustrate the effect of the second advanced start procedure on two test problems, PDS1 and PDS3 in Table 3.4. As can be observed from the table, significant improvement is realized with the advanced start procedure. It is used by default in the LQP algorithm.

Test Problem	Artificial Basis			Advanced Start		
	Sub. time	Mas. time	Total	Sub. time	Mas. time	Total
PDS1	26.33	112.83	139.27	7.10	104.25	111.35
PDS3	251.65	1080.25	1331.9	59.24	929.54	988.99

Table 3.4: Performance of the Advanced Start Procedure.

3.4.5 Performance of the Algorithm

In this section we summarize the performance of the algorithm on a subset of three PDS problems. Table 3.5 provides the statistics with respect to the CPU time spent in the solution of subproblems and master problems of the simplicial decomposition algorithm. Major iterations refer to total the number of penalty minimizations, i.e. executions of Step 2 of the LQP algorithm. The number of vertices retained upon termination of the algorithm and the total number of simplicial decomposition iterations are also reported. The problems PDS1 and PDS3 were also solved using MINOS 5.1. Those runs were performed on an APOLLO DN4000 workstation. PDS5 was not run with MINOS. PDS1 was solved in 341 seconds whereas PDS3 was solved in 81592 seconds.

Problem	Major iters	Simpl. iters	No of vertices	Subprob. time	Master time	Total time
PDS1	7	22	16	7.5	104.18	111.68
PDS3	10	43	33	63.09	943.59	1006.68
PDS5	12	71	56	324.33	6641.33	6966.66

Table 3.5: Performance of the LQP algorithm on a set of three PDS problems.

The objective value upon termination of the algorithm for PDS1 and PDS3 problems agree to the first five digits with the objective values reported by the code MINOS of Murtagh and Saunders [1989]. We note that the objective values at optimality reported in Carolan et.al. [1990], Meyer and Schultz [1990], Marsten et al. [1990], Setiono [1989] only agree to the first two digits of the values reported by the LQP algorithm and MINOS. This led us to conclude that we have slightly different data from the rest of the literature.

3.5 Solving Large Scale Models

We solved all the test problems on the CRAY Y-MP after vectorizing the code. Details of vector computing and parallel decompositions will be presented and discussed in Chapter 5. The results are given in Table 3.6. The number of major iterations is under 15 for all problems and is not reported.

Test Problem	Simpl. iters	No of vertices	Subprob. time	Master time	Total time
PDS1	23	16	1.01	0.85	1.86
PDS3	41	34	12.04	6.73	18.77
PDS5	71	54	55.12	37.97	93.09
PDS10	103	86	232.31	175.82	408.13
PDS15	121	104	559.35	381.08	940.43
PDS20	145	119	1225.83	720.06	1945.89
KEN7	na	na	na	na	na
KEN11	16	9	7.4	8.1	15.5
KEN13	87	14	66.8	70.6	137.5

Table 3.6: Solution times on the CRAY Y-MP.

The problem KEN7 has no binding capacity constraints at the optimal solution and therefore was solved trivially by solving the linear network problem for each commodity. Some of these problems were solved on the same computer by Marsten et al. [1990] using the code OB1 based on interior point methods. The results are summarized in Table 3.7.

Test Problem	Carolan et al.	Marsten et al.	LQP
KEN11	6 mins. 36 secs.	21 secs.	15 secs.
KEN13	20 mins. 45 secs.	1 min. 7 secs.	2 mins. 17 secs.
PDS1			2 secs.
PDS3			19 secs.
PDS5			1 min. 33 secs.
PDS10	3 hrs. 18 mins.	25 mins. 30 secs.	6 mins. 48 secs.
PDS15			15 mins. 40 secs.
PDS20	24 hrs.	4 hrs. 27 mins.	32 mins. 25 secs.

Table 3.7: Comparative solution times of test problems with various methods in CPU hours.

It is clear that the LQP algorithm outperforms substantially the original implementations of the primal-dual interior point methods. of interior point methods. It is, of course, fair to

point out that the interior point algorithms do not exploit the network structure, nor take advantage of the block diagonal sparsity pattern of the constraint matrix. The results of Marsten et al. and GENOS/MC are directly comparable since they were both executed on the same computer and both codes were vectorized.

The same test problems were also solved by Carolan et al. [1990] on the KORBX system using several variants of Karmarkar's algorithm. The comparison between the KORBX system and GENOS/MC does not discriminate between the relative merits of the algorithms and the underlying hardware platform. The comparison is still illuminating since the KORBX system is considered to be one of the first solution technologies for large scale optimization problems. The results by Setiono [1989] were obtained on an Astronautics ZS–1, and the results by Schultz and Meyer [1990] on a Sequent with eleven processors. Details are given in Table 3.8.

Test Problem	Schultz and Meyer	Setiono	LQP
PDS1		29 secs.	2 secs.
PDS3		4 mins. 41 secs.	19 secs.
PDS5	8 mins. 16 secs.	35 mins. 24 secs.	1 min. 33 secs.
PDS10	20 mins. 36 secs.	4 hrs. 52 mins.	6 mins. 48 secs.
PDS15			15 mins. 40 secs.
PDS20	1 hr.		32 mins. 25 secs.

Table 3.8: Comparative solution times of PDS problems with various methods in CPU hours.

3.6 Solving Constrained Network Flow Problems Using the Linear-Quadratic Penalty Technique

It is well documented in the optimization literature that network-structured optimization problems can be solved substantially faster than the general linear program. This observation holds true even with the recent developments of Karmarkar's algorithm [1984] for linear

programming, and the research that followed it on interior point methods. Furthermore, the superior performance of special purpose network algorithms has been documented for both pure and generalized networks, as well as for nonlinear programs. For example, in the mid-seventies, several studies established that codes based on a network simplex algorithm for pure network problems were 150-200 times faster than the state-of-the-art LP codes of the time. See, for example, Glover et al. [1979] and Mulvey [1978]. In the early eighties research concentrated on the generalized network problem. Once more the network simplex algorithm was shown to be approximately 50 times faster than LP codes. See, for instance, Brown and McBride [1985] and Mulvey and Zenios [1985]. This line of research was extended to the nonlinear network problem – see Dembo, Mulvey and Zenios [1989] – for a recent survey. Network specializations of nonlinear programming algorithms — like the primal truncated Newton or simplicial decomposition — were shown to be at least one order of magnitude faster than general purpose nonlinear programming solvers.

Every development in network algorithms was followed by research to use the new algorithms in solving linear programs with large embedded networks. These efforts were generally successful in solving linear programs where the majority of constraints and variables had a network structure. Such programs are known as *networks with side constraints and variables.* In this category are included several well-known classes of problems: the processing (or blending) problem of Koene [1982], the equal flow problem of Ali, Kennington and Shetty [1988], the multicommodity network flow problem, Kennington and Helgason [1980] and so on. The applications of these problems in management science are numerous and well documented in the above references.

In this section we specialize the linear-quadratic penalty algorithm to solve nonlinear networks with side constraints and variables. The primary objective is to test the suitability of the penalty technique to problems with side constraint structures other than multicommodity joint capacity constraints. The technique we propose here can also solve linear network problems with side constraints and side variables. As such it fits in the line of research pursued in the past by several others: McBride [1985], Chen and Engquist [1986], Glover and Klingman [1981], Chen and Saigal [1977] and so on. (See, also, Kennington and Helgason [1980, Chapter 7].) Even in the case of linear networks, however, our approach differs significantly from the earlier studies. Most of the earlier work dealt with specializations of the simplex algorithm. These specializations aimed at developing basis partitioning

techniques that would separate the network basis form the non-network component. These two components were treated using distinct computational procedures. Graph data structures were used to carry out operations on a tree (corresponding to the network basis). General sparse matrix factorizations were applied to the non-network component. When applied to networks with few side constraints these methods were proven very efficient.

3.7 The Linear-Quadratic Penalty Algorithm for Networks with Side Constraints and Variables

In this section we extend the Linear-Quadratic Penalty (LQP) algorithm to solve nonlinear programs with embedded network structures and side variables. Many applications in the operations research literature involve network models with side constraints and side variables. We show in the subsequent sections that the LQP algorithm can efficiently handle this class of problems of which the multicommodity flow problem is a special case. We begin with a formulation of the problem and proceed with the main components of the LQP algorithm.

3.7.1 Problem Formulation

We consider the following side constrained network program:

[SNLP]

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & f(x,z) \\ \text{subject to} \quad & Ax = b \\ & Sx + Pz \leq d \\ & 0 \leq x \leq u \\ & 0 \leq z \leq r \end{aligned}$$

where:

f: $\Re^{n_1+n_2} \rightarrow \Re$ is the objective function, assumed to be convex and continuously differentiable,

$x \in \Re^{n_1}$ is the vector of decision variables which represent flows on a graph,

$z \in \Re^{n_2}$ is the vector of decision variables which represent the side (non-network) columns,

A is an $m \times n_1$ node-arc incidence matrix of some network.

S is the $s \times n_1$ matrix of side (i.e., non-network) constraints imposed on the network flow variables,

P is the $s \times n_2$ matrix of side (i.e., non-network) constraints imposed on the side variables,

$u \in \Re^{n_1}$ are upper bounds on the flow variables x,

$r \in \Re^{n_2}$ are upper bounds on the side variables z,

$b \in \Re^m, d \in \Re^s$ are the right–hand side coefficients of the constraints.

Also, let

$$X = \{(x,z) | Ax = b, 0 \leq x \leq u, 0 \leq z \leq r\}.$$

Throughout the rest of this section, transposition is indicated by a superscript ^{T}, $\nabla_x f$ and $\nabla_z f$ denote the gradient vector of the function f with respect to x and z, and all vectors are column vectors.

The linear-quadratic penalty function is used to eliminate the side constraints by placing those in the objective function. The nonlinear network problem obtained by penalizing the side constraints $Sx + Pz \leq d$ is formulated as:

[NETNLP]

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & \Phi(x,z) = f(x,z) + \mu \sum_j \tilde{p}(\rho_j) \\ \text{subject to} \quad & Ax = b \\ & 0 \leq x \leq u \\ & 0 \leq z \leq r \end{aligned}$$

where $\rho = Sx + Pz - d$, the linear-quadratic penalty function is given by (2.4) and μ is a positive scalar which determines the severity of the penalty. The resulting nonlinear network problem is solved repeatedly with adaptively changing parameters μ and ϵ until suitable stopping criteria are satisfied. The algorithm can be concisely stated as follows:

The Linear–Quadratic Penalty Algorithm

LQP–0 (Initialization.) Find an initial feasible solution to the network component of **SNLP** ignoring the side constraints, i.e., solve the problem

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & f(x,z) \\ \text{subject to} \quad & Ax = b \\ & 0 \leq x \leq u \\ & 0 \leq z \leq r \end{aligned}$$

If the solution to this problem satisfies all side constraints, stop. Otherwise choose initial values for penalty parameters μ and ϵ and go to **LQP–1.**

LQP–1 (Penalty Problem.) Solve – perhaps inexactly – the nonlinear network problem **NETNLP** using a simplicial decomposition algorithm. Go to **LQP–2.**

LQP–2 If the solution satisfies optimality criteria, stop. Otherwise, adjust the penalty parameters μ and ϵ and go to **LQP–1.**

3.7.2 Numerical Experience with Constrained Network Flows

The LQP algorithm was modified to solve problems of the form [**SNLP**]. The code was written in Fortran 77. We refer to the code as the GENOS/LP system. The computational testing was performed on DEC stations 3100 and 5100/200 running Ultrix and for large problems on a CRAY Y-MP. On the DEC stations, the code was compiled with the default compiler optimization option. The algorithm terminates when both of the following error measures are within acceptable tolerance:

1. Absolute error in side constraint feasibility

$$\|Sx + Pz - d\|_\infty \leq \epsilon_{min}$$

2. Bound gap

$$\frac{f(\underline{x},\underline{z}) - V(\hat{u})}{V(\hat{u})} \leq \epsilon_{gap}$$

where $\mathbf{x} = \binom{x}{z}$ is the current iterate and $\binom{\underline{x}}{\underline{z}}$ is obtained from (3.42). The values of ϵ_{min} and ϵ_{gap} used in this study are 10^{-5} and 10^{-2} respectively.

Solving Constrained Matrix Estimation Problems

In this section we report results with constrained versions of two matrix estimation problems from the World Bank. The matrix estimation problem is that of adjusting the entries of Social Accounting Matrices (SAM) for an economy and can be formulated as a nonlinear network optimization problem, see Zenios, Drud and Mulvey [1989]. The first problem SAMKE is a SAM model for Kenya and the second problem SAMBOT is a SAM model for Botswana. Both problems were derived from econometric studies conducted at the World Bank. Both problems have separable objective functions. The problem SAMKE has a weighted entropy objective function of the form $ax \cdot (\log \frac{x}{b} - 1.0)$ for each flow variable and the problem SAMBOT has a quadratic objective function of the form $a \cdot (x-b)^2$. We note that these functions do not satisfy the assumption of Lemma 1 of section 2.5 and therefore we rely on the subproblem lower bound for these problems. Since no initial feasible solution can be readily obtained, we do not compute upper bounds.

We constructed side constraints for these problems as follows. A number of arcs were randomly chosen and a fraction of the sum of the optimal flow values on these arcs was taken as the right-hand side of the inequality. This was repeated as many times as the number of side constraints we added to the problem. Therefore, the side constraints have the following form:

$$\sum_{(ij)\in\mathcal{E}} x_{ij} \geq \beta \sum_{(ij)\in\mathcal{E}} x^*_{ij}$$

where $\mathcal{E}$ is an arbitrary subset of the arcs of the underlying graph and x_{ij} is the flow variable for arc (i,j), x^*_{ij} are the optimal flows obtained by solving the original matrix estimation problem and $\beta \in (0,1]$. Characteristics of the problems are summarized in Table 3.9.

Problem	Network		Optimal
	Nodes	Arcs	value
SAMKE	50	202	−7768.11
SAMBOT	128	662	10.71

Table 3.9: Characteristics of matrix estimation problems.

Starting with one side constraint, both problems were solved with an increasing number of

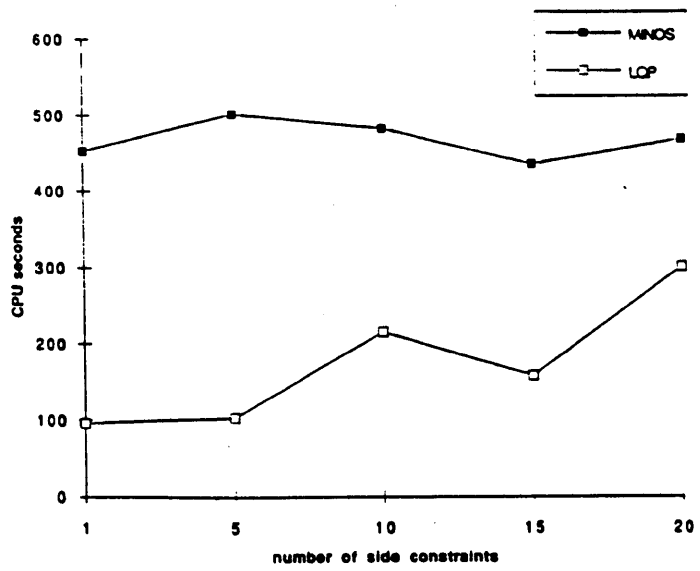


Figure 3.2: Variation of the solution time for constrained matrix estimation problem SAMBOT as a function of the number of side constraints with GENOS/LP and MINOS.

side constraints using the LQP algorithm. Tests were performed on a DEC station 3100. All times are given in CPU seconds. We provide in Figure 3.2 the variation of the CPU time taken by GENOS/LP algorithm to solve SAMBOT and the CPU time taken by MINOS on the same problem as a function of the increasing number of side constraints added to the problem. We observe that while MINOS time does not vary considerably, GENOS/LP outperforms MINOS by a significant margin. However the advantage of GENOS/LP is reduced as the number of side constraints increase. This is not surprising since the LQP method is more sensitive to the size of the network component which gets smaller in percentage as more side constraints are added. In Figure 3.3, we plot the optimal values reported by GENOS/LP and MINOS with the problem SAMBOT as a function of the number of side constraints. In all experiments, GENOS/LP was able to produce reasonably accurate solutions to the problem.

We provide in Table 3.10 a summary of the LQP algorithm statistics for both problems. The problems are referred to as SAMKE25 and SAMBOT20 to indicate the number of side constraints present in the problem.

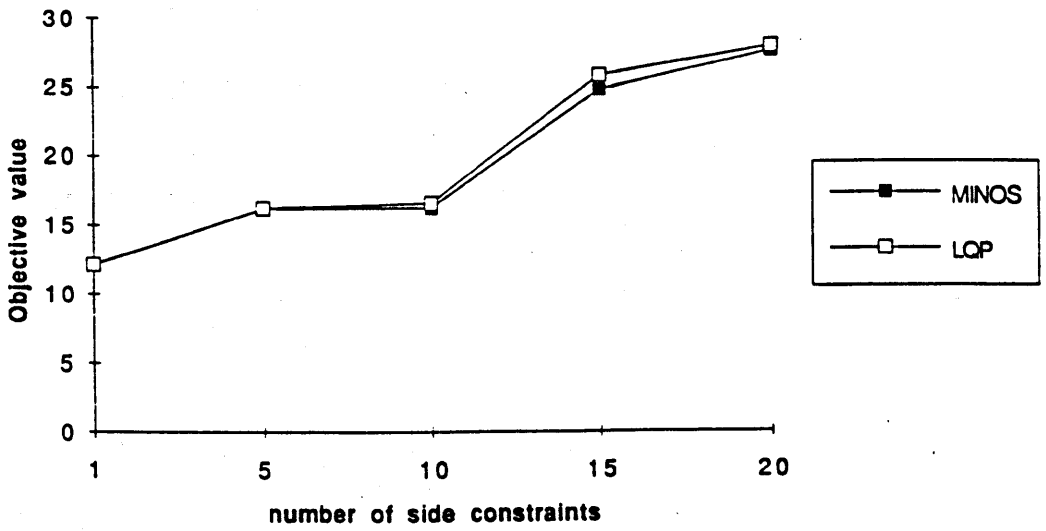


Figure 3.3: Variation of the optimal value for constrained matrix estimation problem SAMBOT as a function of the number of side constraints with GENOS/LP and MINOS.

Problem	Major iters.	Infeasibility	Total time	Objective value	Lower Bound
SAMKE25	3	5.82×10^{-4}	14.	−7678.11	−7624.85
SAMBOT20	7	6×10^{-4}	303.	21.89	28.10

Table 3.10: The LQP statistics with the matrix estimation problems.

The lower bounds in both cases are not very tight. The solution statistics with MINOS for problems SAMKE25 and SAMBOT20 are given in Table 3.11.

Problem	Number of iterations	Optimal value	CPU time
SAMKE25	475	−7630.94	7
SAMBOT20	1804	27.73	472

Table 3.11: Performance of MINOS on the matrix estimation problems.

Although MINOS provides more accurate solutions and was faster with the smaller SAMKE25, the LQP method outperforms MINOS on the larger problem with respect to CPU usage

for different number of side constraints while it produced an acceptable level of accuracy.

Analyzing the NETLIB Test Problems

Using the network extracting heuristics of Bixby and Fourer [1988], we analyzed a subset of NETLIB linear programming test problems. A summary of the characteristics of the test problems and the associated network formulations are given in Table 3.12. As can be observed from the statistics, two of the problems have a large network component to warrant some attention.

Problem	Linear Programming			Network		Side	Side
	Rows	Columns	Opt. Val.	Nodes	Arcs	Const.	Vars.
RECIPE	92	180	-2.6661×10^2	54	140	30	14
GREENBEA	2400	5443	-7.2462×10^7	895	4641	1423	606
GIFFPINC	617	1092	6.9022×10^6	523	1071	69	1
SCAGR25	472	500	-1.4753×10^7	200	372	147	127
SCRS8	491	1169	9.9429×10^2	301	1096	156	78
SHIP12L	1165	5427	1.4701×10^6	735	5321	104	
SIERRA	1228	9252	1.5394×10^7	878	2726	349	
STANDATA	468	3686	1.2576×10^3	96	331	226	789

Table 3.12: The NETLIB problems in LP and network forms.

Our experience with the NETLIB problems using the LQP algorithm revealed that solving these problems as general linear programs is more efficient. We report results with two problems GIFFPINC and SHIP12L. The LQP statistics are given below in Table 3.13. The number of major iterations refer to the number of executions of step 1 of the LQP algorithm. We also report the final objective function value reported on termination and the best lower bound computed thus far. Infeasibility refers to the maximum degree of violation of the side constraints. It was not possible to provide an initial feasible solution for these problems and hence no feasible iterates and upper bounds to optimal value were computed. The tests were performed on a DEC station 3100. SHIP12L was solved in 110 CPU seconds using MINOS and GIFFPINC was soved in 25 seconds using the same code.

Problem	Major iters.	Infeasibility	Total time	Objective value	Lower Bound
GIFFPINC	10	0.242×10^{-2}	8240.9	7.581×10^{6}	6.766×10^{6}
SHIP12L	15	2.37	7200	1.768×10^{6}	NA

Table 3.13: The LQP statistics with the NETLIB problems.

The statistics clearly indicate that these problems proved to be extremely hard for the LQP algorithm. In the case of GIFFPINC, although an acceptable level of infeasibility and a reasonable lower bound was achieved, the objective value is off the known optimal value by a considerable margin. The case of SHIP12L was more problematic. The computation was stopped after two CPU hours and the iterate was still far from reaching the absolute infeasibility tolerance $\epsilon_{min} = 10^{-5}$. It was also impossible to assess the quality of the solutions to nonlinear network (penalty) problems due to the poor quality of the lower bounds. We also note that both problems had a dense side constraint matrix structure, a factor which affects the efficiency of the LQP algorithm negatively.

To conclude, we remark that the LQP technology was not effective in dealing with the NETLIB problems although the experience contributed to the robustness testing of the GENOS/LP code.

3.8 Conclusions

We presented in this chapter a solution method suitable for large scale optimization problems with embedded network structures, and results of extensive computational testing with various test problems. The LQP method is an exterior point method based on a smooth penalty function and can produce feasible iterates if an initial feasible point is available. For applications where several problem instances need to be solved such as the Patient Distribution System there is a high payoff in exploiting the network structure and developing a specialized algorithm. Particularly as the problem size gets bigger, the benefits of using the LQP algorithm become more accentuated. In this chapter we presented strong evidence to support this claim. For smaller problems it is more beneficial to use general purpose algorithms as evidenced by the analysis with the NETLIB problems. Another important factor which affects the performance of the LQP algorithm is the sparsity pattern of the

side constraint matrix. With the PDS and World Bank problems the side constraint matrix had a favourable sparsity pattern whereas the NETLIB problems had a very dense structure. However, the LQP algorithm may still be a viable alternative in smaller problems with relatively few side constraints as observed in the case of constrained matrix estimation problems.

Chapter 4

Alternative Block-Decomposition Techniques for the Nonlinear Network Problem

4.1 Introduction

In Chapter 3 we developed a decomposition algorithm for the solution of network flow problems with side constraints and variables for multicommodity network flow problems. By placing the side constraints into the objective function via a linear-quadratic penalty function we obtained a nonlinear network problem that we solved using the simplicial decomposition algorithm of von Hohenbalken [1977]. In this chapter, we consider alternative decomposition methodologies for the solution of the nonlinear network penalty problem. In the first part of the chapter we focus on a decomposition technique that induces separability by identifying independent search directions in a truncated Newton algorithm specialized for nonlinear one-commodity network flow problems. However the techniques developed here can be easily extended to solve nonlinear multicommodity network flow problems where the coupling occurs in the objective function. An example is the traffic assignment problem. A nonlinear nonseparable multicommodity network flow problem is also produced as a result of placing the mutual capacity constraints into the objective function via a nonlinear nonseparable penalty function. We investigate techniques that would enable a decomposition

of the main computational component of truncated Newton algorithm into independent computations that can be performed in parallel.

In the second part of the chapter we consider a *cyclic* decomposition methodology and study its convergence properties.

Let us recall that by placing the side constraints into the objective function we obtained nonlinear network programs of the form:

[NLNW]

$$\min_{x \in \Re^n} \quad F(x) \tag{4.1}$$

$$\text{s.t.} \quad Ax = \quad b \tag{4.2}$$

$$0 \quad \leq x \leq \quad u \tag{4.3}$$

where $F : \Re^n \longmapsto \Re$ is convex and once continuously differentiable, A is an $m \times n$ node–arc incidence matrix, b, l and $u \in \Re^n$ are given vectors and $x \in \Re^n$ is the vector of decision variables. The node–arc incidence matrix A specifies conservation of flow constraints (2) on some network $G = (N, E)$ with $|N| = m$ and $|E| = n$.

One of the most efficient algorithms for solving large instances of [NLNW] programs is the primal truncated Newton algorithm (PTN) of Dembo and Steihaug [1983], implemented within the active set framework of Murtagh and Saunders [1978]. In this chapter we develop techniques for partitioning Newton's equations — that comprise the primary computational step in PTN — into blocks of equations of reduced size. These blocks of equations can be solved in a fraction of the time required by the original system. Furthermore, the blocks can be solved in parallel both on message passing and shared memory architectures. The block–partitioning techniques exploit the special structure of the basis of network problems. Hence, they can be executed efficiently even for very large problems. The partitioning schemes for pure networks formalize and generalize the procedures designed by Rosenthal [1981] and Escudero [1986a,1986b] for the partitioning of replicated pure network problems. The technique used for partitioning generalized networks extends the scheme proposed in Clark and Meyer [1987] for the solution of linear generalized networks, to nonlinear problems. Some related work on the partitioning of Newton's algorithm for unconstrained optimization is given in Nash and Sofer [1989].

The material presented in this chapter emphasizes three contributions. First, it develops

the block–partitioning methods based on the structure of the network basis. Second, it evaluates the efficacy of the block–partitioned algorithm for solving large scale problems on both serial and parallel computers. Third, it compares the efficiency of these methods with more standard partitioning techniques that do not exploit the basis structure. As a by–product, this study compares two alternative methods for parallel computing within PTN: the block–partitioning developed here and the parallel implementation of Zenios and Mulvey [1988].

With respect to notation: B^T denotes the transpose of matrix B, $B_{.t}$ and $B_{t.}$ denote the t-th column and row respectively of B. With the t-th variable x_t we associate the tuple (i,j) where i is the row with "+1" in the t-th column of A, and j is the row with the arbitrary real value in the t-th column of A. In network terminology, i and j are called the incident nodes of arc (i,j). We will use $t \sim (i,j)$ to indicate this association.

The rest of this chapter is organized as follows: Section 4.2 reviews concepts from PTN and active set methods that are relevant to our work. Section 4.3 reveals the relation between the structure of Newton's equations and the basis of network models and develops the block–partitioning schemes. Section 4.4 discusses implementation issues and reports results from numerical experimentation on serial computers, a shared memory multiprocessor, the Alliant FX/4 and a vector supercomputer the CRAY X–MP/48. Concluding remarks are given in section 4.5. In section 4.6, we discuss a *cyclic* decomposition technique that is also amenable to parallel implementation for the solution of the nonlinear network penalty problem.

4.2 The Truncated Newton Algorithm and Active Sets

The primal truncated Newton algorithm is a feasible direction method within an active set framework. We describe here the algorithm in two steps: First we give a model Newton's algorithm for unconstrained optimization problems. Second, we discuss the active set method that reduces a constrained optimization problem into a sequence of (locally) unconstrained problems in lower dimension. There exists an extensive literature on both techniques. Our main reference is Gill et al. [1981]. The development of active set methods for large scale constrained optimization is given in Murtagh and Saunders [1978]. The truncated Newton algorithm for unconstrained optimization can be found in Dembo and Steihaug [1983]. The

combination of both techniques for pure network problems is given in Dembo [1987] and for generalized networks in Ahlfeld et al. [1987].

4.2.1 Model Truncated Newton Algorithm for Unconstrained Optimization

Consider the unconstrained problem

$$\min_{x \in \Re^n} \quad F(x), \tag{4.4}$$

where $F(x)$ has the same properties as in section refintro. The PTN algorithm starts from an arbitrary feasible point $x^0 \in \Re^n$ and generates a sequence $\{x^k\}$, $k = 1,2,3,\ldots$ such that

$$\lim_{k \to \infty} x_k = x^*,$$

where x^* belongs to the set of optimal solutions to (4.4) (i.e., $x^* \in X^* = \{x | F(x) \leq F(y), \forall\, y \in \Re^n\}$). The iterative step of the algorithm is the following:

$$x^{k+1} = x^k + \alpha^k p^k. \tag{4.5}$$

$\{p^k\}$ is a sequence of descent directions computed by solving the system of (Newton's) equations :

$$\nabla^2 F(x^k) p^k = -\nabla F(x^k) + \eta^k. \tag{4.6}$$

$\nabla^2 F(x^k)$ and $\nabla F(x^k)$ denote the Hessian matrix and gradient vector of $F(x)$ evaluated at point x^k. The sequence η^k is a measure of accuracy in solving equations (4.6). A scale independent measure of the residual error is :

$$r^k = \frac{\|\nabla^2 F(x^{k-1}) p^k + \nabla F(x^{k-1})\|_2}{\|\nabla F(x^{k-1})\|_2}. \tag{4.7}$$

The step direction is computed from (4.6) such that the condition $r^k \leq \eta^k$ is satisfied and the sequence $\{\eta^k\} \to 0$ as $k \to \infty$. $\{\alpha^k\}$ is a sequence of step sizes computed by solving

$$\alpha^k = \arg \min_{\alpha \in \Re^0_+} \{F(x^k + \alpha p^k)\}, \tag{4.8}$$

(i.e., at iteration k the scalar α^k is the step size that minimizes the function $F(x)$ along the direction p^k starting from point x^k). Computing α^k from equation (4.8) corresponds

to an exact minimization calculation that may be expensive for large scale problems. It is possible to use an inexact linesearch. The global convergence of the algorithm is preserved if the step length computed by inexact solution of (4.8) produces a sufficient descent of $F(x)$, satisfying Goldstein–Armijo type conditions.

4.2.2 Model Active Set Algorithm for Constrained Optimization

Consider now the transformation of [NLNW] into a locally unconstrained problem. Following Murtagh and Saunders [1978] we partition the matrix A into the form:

$$A = [\, B \;\; S \;\; N \,]. \tag{4.9}$$

B is a non–singular matrix of dimension $m \times m$ whose columns form a basis. S is a matrix of dimension $m \times r$ and N is a matrix of dimension $m \times (n-m-r)$. We also use $\mathcal{B}$, $\mathcal{S}$ and $\mathcal{N}$ to denote the sets of basic, superbasic and non–basic variables respectively. Similarly we partition x^k into

$$x^k = [\, x_B^k \;\; x_S^k \;\; x_N^k \,]. \tag{4.10}$$

$x_B^k \in \Re^m$ are the basic variables, $x_S^k \in \Re^r$ are the superbasic variables and $x_N^k \in \Re^{n-m-r}$ denote non–basic variables. Non–basic variables — for a given partitioning (4.9)–(4.10)— are kept fixed to one of their bounds. If we now partition the step direction p as

$$p^k = [\, p_B^k \;\; p_S^k \;\; p_N^k \,] \tag{4.11}$$

we require $p_N^k \equiv 0$ (i.e., non–basic variables remain fixed) and furthermore p^k should belong to the nullspace of A (i.e., $Ap^k = 0$), so that p^k is a feasible direction. Hence p^k must satisfy

$$Bp_B^k + Sp_S^k = 0 \text{ or } p_B^k = -(B^{-1}S)p_S^k. \tag{4.12}$$

If the superbasic variables are strictly between their bounds and the basis B is maximal as defined in Dembo and Klincewicz [1985] (i.e., a non–zero step in the basic variables x_B is possible for any choice of direction $(p_B^k \;\; p_S^k \;\; 0)$), then the problem is locally unconstrained with respect to the superbasic variables. Hence a descent direction for p_S^k can be obtained by solving the (projected) Newton's equations:

$$(Z^T \nabla^2 F(x^k) Z) p_S^k = -Z^T \nabla F(x^k) + \eta^k, \tag{4.13}$$

where Z is a basis for the nullspace of A defined as

$$Z = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} \tag{4.14}$$

The primary computational requirement of the algorithm is in solving the system of equations (4.13) of dimension $r \times r$. This system is solved using conjugate gradient with a preconditioner matrix equal to the inverse of of the diagonal of the reduced Hessian matrix $Z^T \nabla^2 F(x) Z$. Calculation of p_B^k from (4.12) involves only a matrix–vector product and is in general an easy computation. The partitioning of the variables and the matrix A into basic, superbasic and non–basic elements is also in general very fast. For some of the bigger problems the solution of system (4.13) takes as much as 99% of the overall solution time. For example, solving problem MULTH2 of Table 4.1 gives rise to a system of equations of dimension 22588×22588. In spite of its very large size, system (4.13) is usually very sparse. In the following section we look at its sparsity pattern and determine ways to partition it in smaller systems that can be solved independently and also on parallel processors.

4.3 Block–partitioning of Newton's Equations

We return now to equation (4.13), and try to identify a partitioning of the matrix $(Z^T \nabla^2 F(x^k) Z)$. Recall that

$$Z = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} \tag{4.15}$$

and assume that the function $F(x) = \sum_{i=1}^{n} F_i(x_i)$ is separable so that the Hessian matrix is diagonal. (This assumption is relaxed in Section 4.3.4). If we ignore momentarily the dense submatrix $(B^{-1}S)$ and assume that

$$Z \doteq \hat{Z} = \begin{bmatrix} I \\ 0 \end{bmatrix} \tag{4.16}$$

(the identity I and null matrix 0 chosen such that Z is conformable to $\nabla^2 F(x^k)$) then the product

$$\hat{Z}^T \nabla^2 F(x^k) \hat{Z}$$

is a matrix of the form diag[H_I 0], where H_I is a diagonal matrix with the t–th diagonal element given by $\frac{\partial^2 F_i(x_t^k)}{\partial x_t^2}$. Hence the complication in partitioning (4.13) is the presence of the submatrix $(B^{-1}S)$. The structure of this submatrix is examined next.

4.3.1 The structure of $(B^{-1}S)$

The matrix B is a basis for the network flows of problem [NLNW]. It is well–known — see, e.g., Dantzig [1963] or Kennington and Helgason [1980]— that the basis of a pure network problem is a lower triangular matrix. The graph associated with this basis matrix is a rooted tree. The basis of a generalized network is characterized by the following theorem(see, e.g., Dantzig [1963, p.421]).

Theorem 1 *Any basis B of a generalized network problem can be put in the form*

$$B = \begin{bmatrix} B^1 & & & & \\ & B^2 & & & \\ & & \cdot & & \\ & & & B^\ell & \\ & & & & \cdot \\ & & & & & B^L \end{bmatrix}$$

where each square submatrix B^ℓ is lower triangular with at most one element above the diagonal.

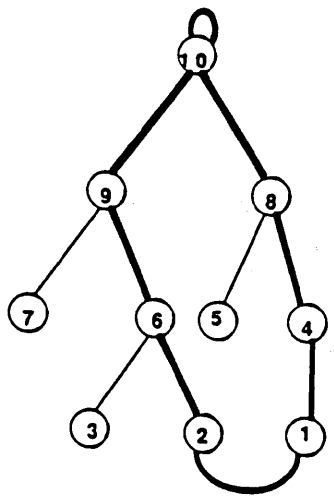
The graph associated with each submatrix B^ℓ is a tree with one additional arc, making it either a rooted tree or a tree with exactly one cycle, and is called a quasi–tree (abbreviated: q–tree). The graph associated with a generalized network basis is a forest of q–trees.

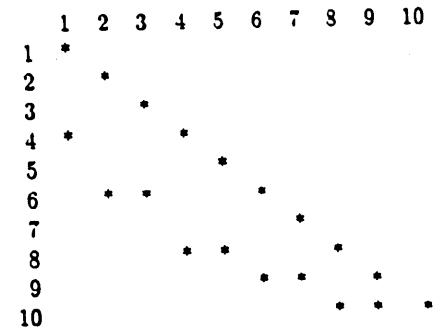
To describe the structure of $(B^{-1}S)$ we first define the *basic–equivalent–paths* (BEP) for a superbasic variable x_t with incident nodes (i,j). For pure network problems it is the set of arcs on the basis tree that lead from node j to node i. The arcs on BEP together with arc $t \sim (i,j)$ create a loop.

In the case of a generalized network, it is the set of arcs that lead from nodes i and j to a cycle; the BEP includes all arcs on the cycle that contains all the nodes. The t–th column of $(B^{-1}S)_{.t}$ has non–zero entries corresponding to the BEP of the t–th superbasic

variable. The numerical values of $(B^{-1}S)$ are ± 1 for pure network problems and arbitrary real numbers for generalized networks; the numerical values are of no consequence to our development. To illustrate the preceding discussion we show in Figure 4.1 the basis of a pure network problem together with the BEP for a superbasic arc and the corresponding column of $(B^{-1}S)$. Figure 4.2 illustrates the same definitions for generalized network problems.

The matrix $(B^{-1}S)$ can be partitioned into submatrices with non–overlapping rows if the columns of each submatrix have BEP with no basic arcs in common with the columns of any other submatrix.





Basis of Pure Network Problem

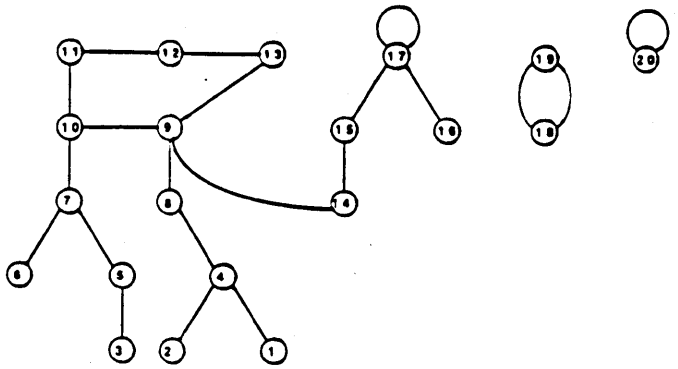
Basic-Equivalent-Path (BEP) for arc with incident nodes (1,2):

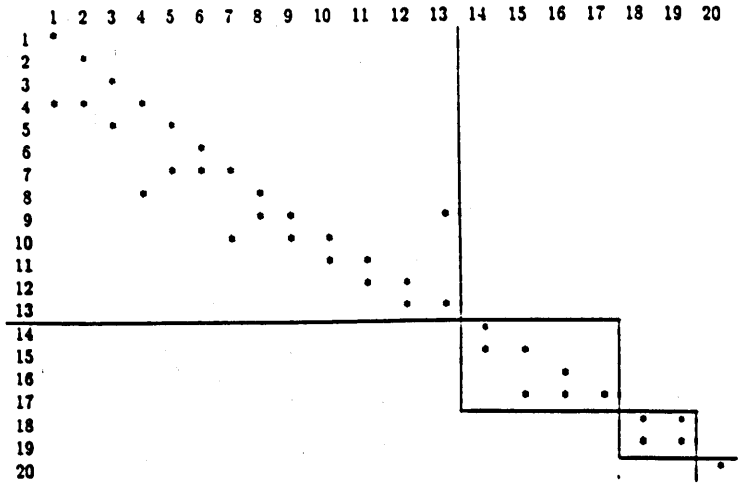
$$\{(2,6),(6,9),(9,10),(10,10),(10,8),(8,4),(4,1)\}.$$

Sparsity pattern of $(B^{-1}S)$ corresponding to superbasic (1,2):

Row no.		Corresponding Basic Arc
1	*	(1,4)
2	*	(2,6)
3	0	
4	*	(4,8)
5	0	
6	*	(6,9)
7	0	
8	*	(8,10)
9	*	(9,10)
10	*	(10,10)

Figure 4.1: Pure network basis: matrix and graph representation, and an example of a basic-equivalent-path.





Basis of Generalized Network Problem

Basic-Equivalent-Path (BEP) for arc (9, 14):

{(9, 10), (10, 11), (11, 12), (12, 5), (13, 9), (14, 15), (15, 17), (17, 17)}.

Sparsity pattern of $B^{-1}S$ corresponding to superbasic arc (9, 14):

Row No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	0	0	0	0	0	0	0	0	•	•	•	•	•	•	•	0	•	0	0	0

Figure 4.2: Generalized network basis: matrix and graph representation, and an example of a basic–equivalent–graph.

4.3.2 Partitioning of $(B^{-1}S)$ for Pure Networks

Let β_t denote an ordered set of binary indices such that $(\beta_t)_l = 1$ if the l–th arc $(i,j) \in \mathcal{B}$ is in the BEP of the t-th arc $(i',j') \in \mathcal{S}$, and $(\beta_t)_l = 0$ otherwise. We seek a partitioning of the set $\mathcal{S}$ into K disjoint independent subsets, say $\mathcal{S}^k$, $k \in \mathcal{K} = \{1,2,\ldots,K\}$ such that

$$\mathcal{S} = \bigcup_{k=1}^{K} \mathcal{S}^k \tag{4.17}$$

$$\text{and } t \in \mathcal{S}^{k_1} \quad \text{and} \quad u \in \mathcal{S}^{k_2} \quad \text{iff} \quad \beta_t \vee \beta_u = 0 \; \forall \, k_1 \neq k_2 \in \mathcal{K} \tag{4.18}$$

(i.e., the sets β_t and β_u have no overlapping non–zeroes, and hence there is no common basic arc in the BEP of t–th and u–th superbasic arcs).

Escudero [1986b] was the first one to propose the partitioning of $\mathcal{S}$ into independent superbasic subsets $\mathcal{S}^k$ according to equation (4.17)–(18), for replicated pure networks. These replicated networks consist of subnetworks with identical structure and are connected by linking arcs. (These linking arcs represent inventory flow for his problems that are multiperiod networks.) In the same reference Escudero gives a procedure for identifying the independent superbasic sets $\mathcal{S}^k$.

In this section we formulate the problem of identifying independent superbasic sets as problems from graph theory.

Define the graph $G_{\mathcal{S}} = \{\mathcal{V}, \mathcal{E}_{\mathcal{S}}\}$ where the node and edge sets are defined as $\mathcal{V} = \{1,2,3,\ldots,|\mathcal{S}|\}$ and $\mathcal{E}_{\mathcal{S}} = \{(t,u) | t,u \in \mathcal{V}, \beta_t \vee \beta_u \neq 0\}$. A node is associated with each superbasic variable $t \in \mathcal{S}$, and an edge is incident to two nodes if and only if the BEP of the corresponding superbasic variables overlap. We call $G_{\mathcal{S}}$ the *connectivity graph* of the superbasic set S. It can be constructed as the adjacency graph of the matrix $Z^T Z$.

The first partitioning scheme simply formalizes Escudero's procedures.

Partitioning Scheme I: Find the connected components of $G_{\mathcal{S}}$, say $\mathcal{V}^k \subseteq \mathcal{V}$, $k = 1,2,\ldots,K$. Then $\mathcal{S}^k = \{t | t \in \mathcal{V}^k\}$ will satisfy conditions (4.17)–(18) – by definition of connected components – and hence $\mathcal{S}^k$, $k = 1,2,\ldots,K$ are independent superbasic sets. Finding connected components of $G_{\mathcal{S}}$ can be achieved with the algorithm of Tarjan [1972]. Unfortunately it is not always possible to find more than one connected component of $G_{\mathcal{S}}$. For the case of replicated networks such a partitioning usually exists since the interaction among superbasics in different subnetworks is weak. This explains the success of Escudero's method for the multiperiod, hydroelectric power

scheduling networks. The second partitioning scheme we propose here allows us to partition the superbasic set $\mathcal{S}$ for a broader class of problems.

Partitioning Scheme II: Find the articulation points of $G_{\mathcal{S}}$. Let $\mathcal{C} \subset \mathcal{V}$ be the set of articulation points and $\mathcal{S}_{\mathcal{C}}$ the set of superbasic variables corresponding to the set $\mathcal{C}$. Update the set of superbasic variables by $\mathcal{S}' = \mathcal{S} \setminus \mathcal{C}$ and let $\mathcal{N} = \mathcal{N} \cup \mathcal{C}$. The new superbasic set S' now consists of at least $|\mathcal{C}|$ independent subsets.

The fact that identifying articulation points leads to partitioning of the superbasic sets can easily be seen by the definition of articulation points (see, e.g., Aho, Hopcroft and Ullman [1974]). A polynomial algorithm for finding articulation points of a graph is given in Tarjan [1972], and was used in our implementation of this partitioning scheme.

4.3.3 Partitioning of $(B^{-1}S)$ for Generalized Networks

The graph partitioning schemes discussed in section 4.3.2 for pure network problems can also be applied in the case of generalized networks. We will refer to these procedures as scheme GN–I (for generalized networks–I). Here we develop alternative techniques to partition the superbasic set of generalized network problems that take advantage of the block structure of the generalized network basis.

In section 4.3.1 we observed that the graph associated with the basis of a generalized network problem is a collection of quasi–trees. Suppose the basis matrix B consists of submatrices B^l, $i = 1, \ldots, L$. We denote the graph(quasi–tree) associated with B^l by $G_l = (N_l, E_l)$. The superbasic set $\mathcal{S}$ can be partitioned in subsets $\mathcal{S}^k$ defined by

$$\mathcal{S}^k = \{(i,j) \in \mathcal{S} | i,j \in N_l\} \; \forall \, l = 1,2,\ldots,L \tag{4.19}$$

with $\cup_{k=1}^{L} \mathcal{S}^k \subseteq \mathcal{S}$. This partitioning scheme will ignore any superbasic variables that connect basis submatrices. A partitioning scheme that includes additional superbasic variables is the following: Given indices k, $p(k) \leq l$ and $q(k) \leq l$, $p(k) \neq q(k)$ choose $B^{p(k)}$ and $B^{q(k)}$ and define

$$\mathcal{S}^{p_k q_k} = \{(i,j) \in \mathcal{S} | i \in N_{p(k)}, j \in N_{q(k)}\} \tag{4.20}$$

$$\mathcal{S}^{p_k} = \{(i,j) \in \mathcal{S} | i,j \in N_{p(k)}\} \tag{4.21}$$

$$\mathcal{S}^{q_k} = \{(i,j) \in \mathcal{S} | i,j \in N_{q(k)}\} \tag{4.22}$$

and finally $\mathcal{S}^k = \mathcal{S}^{p_k q_k} \cup \mathcal{S}^{p_k} \cup \mathcal{S}^{q_k}$. To ensure that two set $\mathcal{S}^{k_1}, \mathcal{S}^{k_2}$ are independent we require

$$B^{p(k_1)} \neq B^{p(k_2)} \neq B^{q(k_2)}$$

$$B^{q(k_1)} \neq B^{p(k_2)} \neq B^{q(k_2)}.$$

We now describe a procedure that we have found to work well to identify these independent subsets $\mathcal{S}^k$ of superbasic arcs. The procedure works as follows: at each iteration, a pair of basis subgraphs connected with the largest number of superbasic arcs is identified, these superbasic arcs together with the superbasic arcs that connect nodes in each subgraph form a subset $\mathcal{S}^k$. To ensure the independence of the subsets thus formed, the basis subgraphs used in constructing the subsets are marked and not considered in subsequent iterations. A formal statement of the procedure is given below.

$\mathcal{U} = \{1, 2, \ldots, l\}$; $k = 0$; $scount = 0$; $\rho \in [0.5, 1)$

Repeat until $scount \geq \rho|\mathcal{S}|$ or $|\mathcal{U}| \leq 1$

1. $k = k + 1$
2. Find $r(k)$ and $s(k)$ such that $r(k) \neq s(k)$ and $|\mathcal{S}_1^{r(k)s(k)}| = \max_{p,q \in \mathcal{U}} |\mathcal{S}_1^{pq}|$
3. $scount = scount + |\mathcal{S}^{r(k)s(k)}| + |\mathcal{S}^{r(k)}| + |\mathcal{S}^{s(k)}|$
4. $\mathcal{S}^k = \mathcal{S}_1^{rs} \cup \mathcal{S}_2^{rs} \cup \mathcal{S}_3^{rs}$
5. $\mathcal{U} = \mathcal{U} \setminus \{r, s\}$

We will refer to the procedure described above as partitioning scheme GN–II. We note that scheme GN–II may place some candidate superbasic arcs into the nonbasic set when creating the independent subsets $\mathcal{S}^k$. It is also advisable to choose ρ as close to 1 as possible. Some preliminary experimentation is needed to find a suitable value of ρ in some cases. We also remark that alternative procedures which take into account the interaction of more than two basis subgraphs can be designed. However this scheme achieved a satisfactory partitioning of the superbasic set with the generalized networks problems used in this study.

To illustrate the partitioning procedure on Figure 4.2, let us assume that the superbasic set is as follows:
$S = \{(6,11),(3,8),(9,14),(14,17),(13,19),(7,19),(1,18),(16,20),(18,20),$
$(14,16),(17,20)\}$. Scheme GN-II will identify $S^1 = \{(1,18),(7,19),(13,19),(6,11),(3,8)\}$ and $S^2 = \{(14,17),(14,16),(16,20),(17,20)\}$. In this example, superbasic arcs $(9,14),(16,20)$ are placed into the nonbasic set to ensure independence of S^1 and S^2.

Test problem	Size	No. of Basis submatrices	Free arcs at opt.	Obj. value
MULTA4	400–1002	9–10	602	0.32713×10^7
MULTA8	800–2000	15–22	1200	0.66535×10^7
MULTA12	1200–3603	23–27	2403	0.10615×10^8
MULTB4	2000–4010	8–31	2010	0.130059×10^7
MULTB8	4000–8022	21–28	4022	0.267280×10^8
MULTB12	6000–15039	35–36	9039	0.175176×10^8
MULTB15	7500–18000	40–50	10500	0.3779514×10^7
MULTC2	2000–5008	4–5	3008	0.53512×10^7
MULTC4	4000–10027	5–8	6027	0.111013×10^8
MULTC8	8000–20047	40–50	12047	0.201423×10^8
MULTH1	11000–27571	40–60	16571	0.566242×10^7
MULTH2	15000–37588	50–60	22588	0.795584×10^7
PTN150	150–196	1	44	-0.481973×10^5
PTN660	666–906	1	240	-0.206107×10^6
STICK1	209–454	1	246	6.934392
STICK2	650-1412	1	763	3.124563
STICK3	782-1686	1	905	0.111797×10^2
STICK4	832-2264	1	1433	1.566195

Table 4.1 : Test Problem Characteristics.

4.3.4 Extensions to Non–Separable Problems

We can develop partitioning of the system $(Z^T\nabla^2 F(x)Z)$ for the case when the function $F(x)$ is non–separable. In this case the Hessian matrix is not diagonal. The partitioning condition (4.17)–(18) from section 4.3.2 has to be modified as follows. Let $t, u \in \mathcal{S}$. Then

$$t \in \mathcal{S}^{k_1} \quad \text{and} \quad u \in \mathcal{S}^{k_2} \quad \text{iff} \quad \beta_t \vee \beta_u = 0 \quad \text{and} \quad \frac{\partial^2 F(x)}{\partial x_t x_u} = 0 \tag{4.23}$$

The connectivity graph associated with the superbasic set $\mathcal{S}$ is now defined as the graph $G_{\mathcal{S}} = \{\mathcal{V}, \mathcal{E}_{\mathcal{S}}\}$ where $\mathcal{V} = \{1, 2, 3, \ldots, |\mathcal{S}|\}$ – as in section 4.3.2 – and $\mathcal{E}_{\mathcal{S}} = \{(t,u) |\ t, u \in \mathcal{V},\ \beta_t \vee \beta_u \neq 0 \text{ and } \frac{\partial^2 F(x)}{\partial x_t x_u} \neq 0\}$ (i.e., an edge is incident to two nodes iff the BEP of the corresponding overlap and changes in the value of one variable, x_t, change the objective value for the second, x_u). The connectivity graph can be obtained from the adjacency graph of $(Z^T\nabla^2 F(x)Z)$. With this definition of connectivity graph, we can now apply either partitioning scheme from section 4.3.2. Similarly we can extend the partitioning technique of section 4.3.3 to handle nonseparable generalized network problems as well.

4.4 Computational Experiments

The partitioning techniques discussed earlier were implemented in the network optimizer GENOS of Mulvey and Zenios [1987].[1] The modified code, which we call GENOS/PCG, was used to solve a collection of nonlinear problems, both pure and generalized. (PCG stands for Partitioned Conjugate Gradient, since GENOS is using conjugate gradient to solve the partitioned systems of Newton's equations). The objective of the numerical experiments is to establish the performance of PTN when Newton's equations are solved in block–partitioned form. Furthermore, we solved the test problems on a shared memory vector multiprocessor to study the performance of the block–partitioned PTN with parallel computing. Finally we compare the performance of the parallel implementation of GENOS/PCG to an alternative parallel implementation proposed by Zenios and Mulvey [1988].

[1] GENOS is a collection of algorithms for solving network problems, including network simplex, primal truncated Newton and simplicial decomposition. The partitioning techniques were implemented within the primal truncated Newton solver. Details on the implementation of this solver within GENOS using sparse graph data structures are given in Ahlfeld et.al. [1987].

GENOS and the modifications in GENOS/PCG are written in Fortran 77. Experiments on serial computer were carried out on a VAX 8700 at the Wharton School, running VMS. The programs were compiled with the default compiler optimization option. Parallel computing experiments were carried out on an Alliant FX/4 of the HERMES Laboratory for Financial Modeling and Simulation at the Wharton School, running Ultrix. The level of optimization used was at least $-Og$ (i.e., global scalar optimizations). The flag $-O$ was used for experiments using the vector and parallel features of the Alliant. All times are reported in CPU seconds, exclusive of input and output. The termination tolerance η^k is adjusted dynamically using a forcing sequence; its final value is 10^{-2}.

Problem	Major Iter.	Subspace dimension	Number of artic. points	Number of Comp.	Sizes of Components.	Partition. time
PTN150						0.53
	1	45	4	4	3-2-4-36	
	2	43	2	4	2-1-34 -3	
	3	44	2	4	2-1-35-3	
PTN660						0.68
	1	236	5	6	2-2-2-2-1-224	
	2	238	4	5	2-2-2-2-226	
STICK1						0.33
	1	246	7	8	24-6-1-1-23-3-8-107	
	2	246	11	12	14-10-6-22-105-8-2	
	3	246	8	9	14-10-5-131-7-3	
STICK2						0.66
	1	763	11	12	5-4-590-11-8	
	2	763	14	15	517-32-31-11-9	
STICK3						0.27
	1	905	10	11	124-237-19-56-12	
	2	905	9	10	247-180-12-11-9	
STICK4						1.41
	1	1433	9	10	1341-28-26-12-2-2	

Table 4.2: **Articulation point analysis: Results for PTN and STICK problems.**

4.4.1 Test Problems

The characteristics of the test problems are summarized in Table 4.1. In addition to the size of the problems we give the number of superbasic arcs at optimality; this is the dimension of the system of equations we had to solve at the last iteration of the algorithm. The MULTxn problems were generated by the network problem generator of Chang and Engquist [1986]. They are multiperiod networks: a basic generalized network structure is replicated for several time periods and inventory–type links connect the replicated components. All problems have a quadratic objective function of the form $\sum_t a_t x_t^2$ with $a_[t] \in [1, 100]$. The number at the end of each problem name indicates the number of replications. For example, MULTC2 is a 2–period model; each single period network has 1000 nodes. MULTC8 is a 8–period model with a total of 8000 nodes. The STICKn and PTNn problems were obtained from Ahlfeld et al. [1987].

4.4.2 Solving Pure Network Problems

We implemented both partitioning schemes I and II discussed in section 4.3.2. First, we need to construct the connectivity graph G_S. This is the adjacency graph of the matrix $Z^T Z$. Due to the large size of the matrix Z it is much more efficient to construct the connectivity graph G_S based on the structural non–zeros of Z than to form the product $Z^T Z$. For example finding the adjacency graph of of $Z^T Z$ takes 21.6 seconds for PTN660 and 18.5 seconds for STICK1. Working on the matrix Z instead, the same graphs are constructed in 4.1 and 0.8 seconds respectively. The most efficient implementation was adopted in all experiments.

Partitioning scheme I was implemented using an algorithm for connected components due to Tarjan [1972]. For the test problems we have available the connectivity graphs G_S are very dense and in all cases there is only one connected component.

Partitioning scheme II was implemented using an articulation point algorithm due to Tarjan [1972]. This algorithm was then applied to the connectivity graph of the superbasic set for the pure network problems STICK1–4 and PTN150–660 at each major iteration of GENOS/PCG. The results are summarized in Table 4.2. Under the column "Size of components" we do not list the (usually large) number of components corresponding to a single superbasic variable. The difference between the size of the subspace and the total of

the sizes of the disconnected components is the number of single–variable sets. We give the total CPU seconds spent in the graph partitioning procedure during the execution of the primal truncated Newton algorithm under the heading "Partitioning time".

We observe from the table that these test problems do not produce independent superbasic sets of (approximately) equal sizes. The largest independent set dominates the computations and solution time for the block–partitioned Newton's equations is only marginally better than the solution time required in solving the equations over the original subspace. In addition some overhead is incurred in the creating the connectivity graph and the subsequent partitioning. The connectivity graph of the superbasic sets tend to be dense, and hence very large. For example the connectivity graph for the superbasic set of PTN660 has 236 nodes and 5768 arcs. The articulation point algorithm identified 4 articulation points in 0.1 seconds. However creating the connectivity graph from the matrix $(B^{-1}S)$ takes 0.48 seconds.

We conclude that for pure network problems the block–partitioning techniques do not offer any computational savings. However, for problems that have additional structures – like Escudero's multiperiod networks – then the partitioning could be very effective. Nevertheless, Since an attempt to partition the network can be executed very efficiently, it is included as an optional preprocessing phase in GENOS/PCG.

	GENOS					GENOS/PCG					Ratio
Problem	PTN	SB	CG	LS	Total	PTN	SB	CG	LS	Total	
MULTA4	2.32	2.20	35.38	0.50	40.42	2.23	1.83	9.64	1.57	15.27	2.64
MULTA8	4.83	4.39	112.47	1.03	122.73	6.72	5.26	41.21	6.85	60.00	2.04
MULTA12	7.94	8.97	241.58	1.86	260.35	13.35	11.70	77.49	12.85	115.47	2.25
MULTB4	14.60	14.04	506.27	2.73	537.64	22.66	16.69	228.32	12.37	280.0	1.92
MULTB8	29.00	29.69	1227.99	4.48	1291.16	64.92	45.56	601.06	28.42	739.97	1.74
MULTB12	57.91	60.16	3130.59	9.73	3258.40	91.98	85.00	1537.10	72.95	1787	1.82
MULTB15	96.41	97.44	15092.50	17.32	15303.72	114.35	101.87	4398.57	98.41	4713.20	3.24
MULTC2	21.41	29.25	696.56	2.50	749.74	27.51	29.25	557.90	7.24	621.99	1.20
MULTC4	43.7	52.33	2233.40	6.95	2336.40	83.51	65.85	1372.41	22.54	1544.00	1.51
MULTC8	123.94	106.70	21659.00	14.99	21904.70	214.86	146.41	3666.11	144.23	4171.62	5.25
MULTH1	102.29	100.88	29908.23	14.54	30125.95	186.56	163.82	6425.61	161.08	6937.08	4.34
MULTH2	226.37	205.33	80902.46	40.59	81374.76	352.59	265.59	26009.06	350.60	26977.00	3.03

Table 4.3 : Solution times with GENOS and GENOS/PCG on the VAX 8700

4.4.3 Solving Generalized Network Problems

We implemented both partitioning schemes GN-I and GN–II. We observe that the scheme GN–II based on the structure of the basis graph is more efficient and produces better partitioning of the superbasic set $\mathcal{S}$. For example, problem MULTA4 is solved by GENOS in 40 seconds without partitioning. Partitioning scheme GN-I takes 54 seconds whereas with partitioning scheme B the entire problem is solved in 15 seconds. We adopted the scheme GN–IIB in solving all the test problems, as shown in Table 4.3 together with the results from GENOS. [2] The speedup factor indicates the ratio of total solution times of GENOS by that of GENOS/PCG. For smaller problems where the network basis does not partition evenly, the savings are reduced due to overhead in creating the subspaces. Improvements in performance increas with the problem size. We observe an improvement in performance by a factor of 1.20 to 5.25.

One final observation on these experiments. We noted earlier that with the partitioning procedure described in section 4.3.3 some superbasic arcs may be placed in the nonbasic set in order to achieve independence of the blocks. This usually affects the accuracy of the solution. However, as the optimality tolerance is sufficiently decreased, one iteration without partitioning the superbasic set is sufficient to achieve high accuracy. But this step is computationally expensive, particularly for larger problems. Ignoring this step results in significant savings in performance. The objective value at termination in this case is within 0.5% from the objective value given by GENOS. This observation is illustrated in Figure 4.3. In all experiments the last step is performed and hence the problem is solved to optimality.

4.4.4 Parallel Implementation

Solving Newton's Equations. First we tested the parallel implementation of GENOS/PCG in solving the set of blocks in a particular instance of Newton's equation. Using problem MULTB12 we isolated one block of dimension 750×750 replicated it 20 times and solved it in parallel (i.e., we assumed that the system of equations in the case of MULTB12 would

[2] PTN indicates the overhead in forming the network basis and other initializations, SB stands for subspace selection, i.e., the selection of the superbasic variables. This is the step where partitioning schemes A and B are incorporated into the program. CG and LS stand for the conjugate gradient and linesearch procedures respectively.

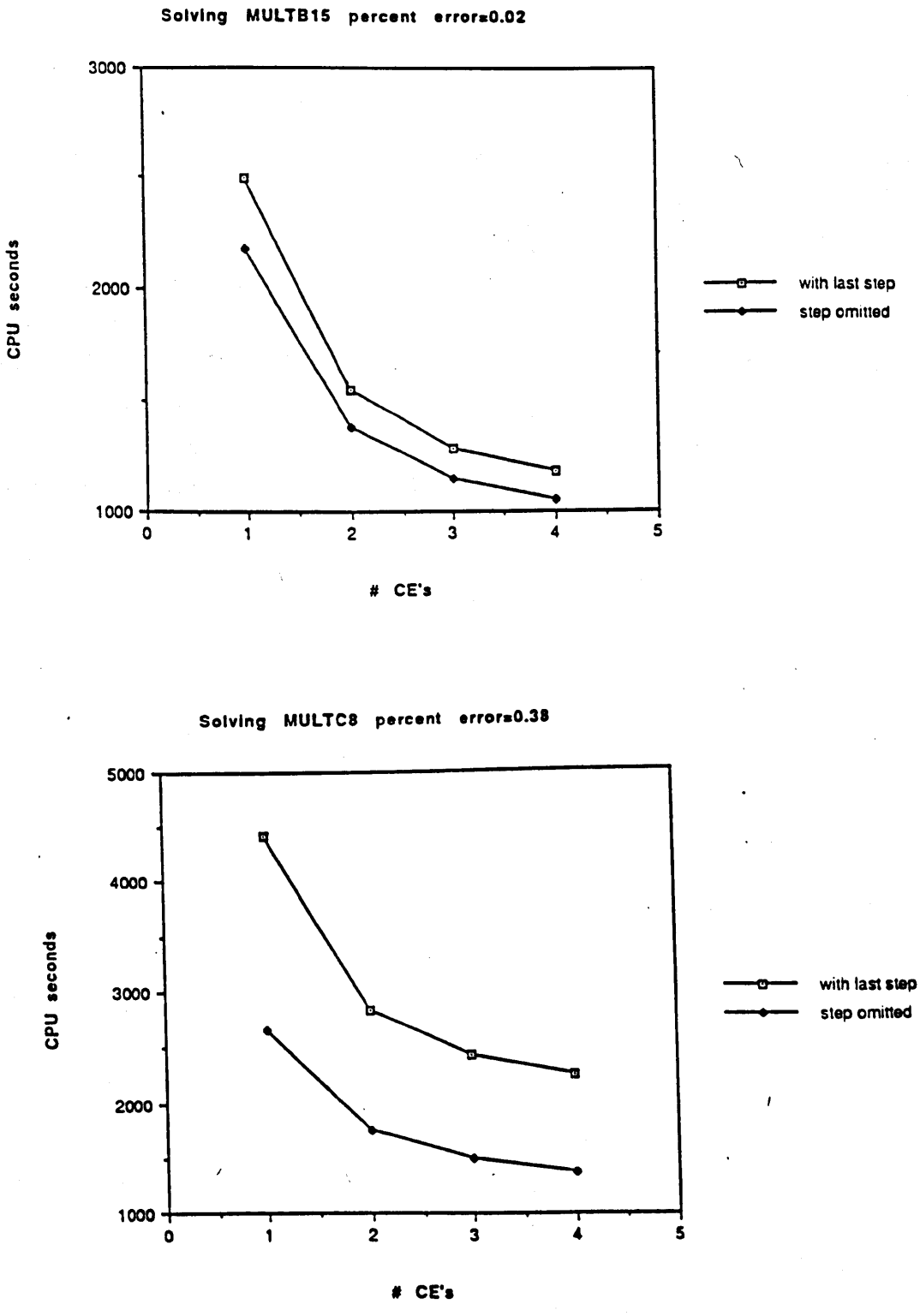


Figure 4.3: Solution times with and without the superbasic partitioning step at the last iteration of PTN.

partition into 20 blocks of equal size. This would be the ideal situation.) Figure 4.4 shows the solution time when solving this set of 20 blocks using 1–4 processors. Significant speedup factors are observed when using parallel processors ranging from 1.6 on 2 CE's to 3.4 on 4. This is due to the overhead incurred in concurrent subroutine calls. In Figure 4.4 we also show the solution times when solving the actual set of 20 blocks —with dimensions ranging from 10×10 to 900×900. While we still observe significant speedup between the serial and parallel implementation the speedup factors are lower than those observed in the previous experiment. The difference is attributed to the uneven load balancing among processors that is due to the varying sizes of the blocks.

Test problem	GENOS (1CE)	GENOS/PCG (4CE)	Speedup
MULTB12	2316.00	753.68	3.07
MULTB15	6362.59	1172.81	5.42
MULTC8	6723.11	2245.15	2.99
MULTH1	17045.85	2009.73	8.48
MULTH2	25336.70	3982.50	6.35

Table 4.4: Comparing serial GENOS with parallel GENOS/PCG.

The Linesearch Procedure. Parallel CE's were also used in the linesearch procedure of GENOS/PCG. The linesearch routine of GENOS (see Ahlfeld et al. [1987]) is a quadratic interpolation with safeguards. At every iteration of the linesearch algorithm the function value, gradient vector and Hessian matrix are evaluated for all the arcs. In GENOS/PCG we need only evaluate this information for those basic and superbasic variables that appear in the current block. Furthermore multiple processors can compute in parallel the required information for multiple arcs in the block. Figure 4.5 illustrates the speedup factor of both the routine that provides function, gradient and Hessian values as well as the speedup of the overall linesearch procedure.

Comparing GENOS with Parallel GENOS/PCG

As a concluding test on the Alliant we run GENOS on 1 Computational Element (CE) of the Alliant with the fully parallellized GENOS/PCG running on 4 CEs. The observed speedup factor for some of the bigger problems is shown in Table 4.4. The speedup factor in most cases exceeds 4 which is the number of parallel CEs used. This is due to the combined effect of solving a sequence of smaller problems and the effect of multiprocessing. These two effects were analyzed separately in the results of Table 4.3 and in section 4.4.4 respectively.

4.4.5 Comparison with Alternative Parallel Implementations

Zenios and Mulvey [1988] proposed an alternative parallel implementation of PTN based on the row-wise distribution of the nullspace matrix among processors. This implementation produced significant speedups when implemented on the CRAY X-MP. We tested the

same implementation on the Alliant FX/4. Hence, we now have two alternative parallel implementations on two different architectures and can draw some conclusions on the relative merits of the two methods. The following diagram indicates which method should be preferred for different problem structures and computer systems;(MPTN indicates the microtasked implementation of Zenios and Mulvey, BPTN indicates the block–partitioned methods developed here).

Problem Structure		Tightly Coupled Multiprocessor	Loosely Coupled Multiprocessor
Pure	Poor Partitioning	MPTN	MPTN
Networks	Good partitioning	MPTN	BPTN
Generalized Networks		MPTN	BPTN

The advantage of MPTN is that its implementation does not require any additional computing, and it results into even load balancing among processors. The disadvantage is that the granularity of the parallel tasks is very small; the overhead in spawning tasks on some computers could be significant compared to the amount of computation performed by the task.

The advantage of BPTN is that it produces tasks of large granularity. The disadvantage is that it may produce uneven load balancing among processors. Furthermore, BPTN need to execute the partitioning algorithms and this overhead can add significantly to the total solution time.

Hence, for tightly coupled systems where the overhead of spawning a task is only a few machine cycles — as in the case of CRAY X–MP — MPTN should be preferred. From Figure 4.6 we observed that MPTN achieves a speedup of 2.6 on 3 CPU's for both pure and generalized networks. BPTN does not achieve any speedups for the pure network test problems and a speedup of 1.41 for generalized networks.

For more loosely coupled systems, that may also include distributed memory architec-

Test problem	GENOS	GENOS/PCG
MULTB8	101.92	49.90
MULTB12	179.36	110.79
MULTB15	795.29	291.64
MULTC4	141.05	102.23
MULTC8	869.66	531.18

Table 4.5: Comparing GENOS with GENOS/PCG on the CRAY X-MP/48.

tures, the BPTN is preferred — unless the problems do not partition well. For example MPTN achieves a speedup of 1.75 on a 4 processor Alliant FX/4. BPTN achieves a speedup of 2.21 on the same system for generalized network problems that partition well.

4.4.6 Solving the Test Problems on a CRAY X-MP/48

Although we were able to achieve significant improvements both on sequential and parallel implementations we observed that solving the multiperiod problems was still taking a considerable amount of time. As a final test we conducted some experiments on the CRAY X-MP/48 to test the effectiveness of the partitioning schemes on a different parallel architecture. The results are summarized in Table 4.5. The tests were conducted with the default vector option of the CRAY FORTRAN compiler. The results are stated in CPU seconds. As can be observed from the table modest gains were possible with the vector supercomputer CRAY X-MP/48.

4.5 Concluding Observations

We developed here techniques for partitioning Newton's equations in the context of solving nonlinear network problems. The techniques appear to be quite effective and efficient for generalized network problems and are also well suited for parallel computations. In Figure 4.7 we use our generalized network problems to illustrate the combined effect of both the partitioning schemes and the computer architecture. As observed in the figure the reduction in the solution time increases as the problem size gets larger. The partitioning techniques can also be applied efficiently for pure network problems. However our current

collection of pure network test problems does not partition well.

Our study also provides guidelines on choosing among two alternative parallel implementations depending on characteristics of the problem and the parallel computing platform.

We finally remark that the partitioning techniques remain applicable in the case of multicommodity network flow problems. In this respect, the analysis of section 4.3.4 are directly applicable. By placing the side constraints into the objective function via a penalty term, the resulting problem is a nonlinear problem with block network constraints. However empirical investigation in this direction is not pursued as part of this dissertation research.

4.6 A Cyclic Decomposition Technique for the Solution of the Penalty Problem

Let us recall the penalty problem

$$\min_{x \in X} \Phi_{\mu,\epsilon}(x) = f(x) + \mu \sum_{j=1}^{s} \tilde{p}(y_j)$$

where $y_j = (Ex - d)_j$, for $j = 1, 2, \ldots, s$ and μ is a positive real number and X is a Cartesian product set. We used a linearization technique to induce separability in $\Phi_{\mu,\epsilon}$ to take advantage of the Cartesian product structure of the constraint set in the multicommodity network flow problem. The question we wish to answer in this section is the following: is it possible to decompose the problem without using a linearization based technique? We present some preliminary work undertaken to answer this question.

Let us simply denote the function $\Phi_{\mu,\epsilon}$ as Φ for ease of notation since we are concerned about finding the solution of the penalty problem for given parameters μ and ϵ which will be kept constant during the minimization. Suppose, as in Chapter 3, that we have K commodities. We term *coordinate direction* iterations the steps taken to solve the penalty problem and use an index t to indicate the *age* of the iterate x. A single coordinate direction iteration enables us to get $x(t+1)$ from $x(t)$ where

$$x(t) = (x_1(t), x_2(t), \ldots, x_K(t))$$

where the component $x_i(t)$ corresponds to block (i.e. commodity) i. Therefore we have the iteration

$$x_i(t+1) = \arg\min_{x_i \in X_i} \Phi(x_{-i}(t), \mu) \quad i = 1, \ldots, K \tag{4.24}$$

where

$$x_{-i}(t) \stackrel{\text{def}}{=} (x_1(t+1), \ldots, x_{i-1}(t+1), x_i, x_{i+1}(t), \ldots, x_K(t)),$$

X_i denotes the feasible region for commodity i, i.e.

$$X_i = \{x_i | A_i x_i = b_i, 0 \leq x_i \leq u_i\},$$

and A_i denotes the node-arc incidence matrix of commodity i. The remaining entities are defined as usual. To analyze the convergence of the iterations, we consider the following nonlinear Gauss–Seidel algorithm and a related result from Bertsekas and Tsitsiklis [1989]. Their convexity assumptions can be weakened somewhat as we show in the proof of the result. Consider

$$x_i(t+1) = \arg \min_{x_i \in X_i} \Phi(x_1(t+1), \ldots, x_{i-1}(t+1), x_i, x_{i+1}(t), \ldots, x_K(t)) \qquad (4.25)$$

where each X_i is a compact and convex subset of $\Re^{n_i}$ and X is the Cartesian product of the sets X_i.

Proposition 1 *Suppose that $\Phi : \Re^n \to \Re$ is continuously differentiable and pseudoconvex on the set X. Furthermore suppose that for each i, Φ is a strictly quasiconvex function of x^i, when the values of the other components of x are held constant. Let $\{x(t)\}$ be the sequence generated by the nonlinear Gauss-Seidel algorithm, assumed to be well-defined. Then every limit point of $\{x(t)\}$ minimizes Φ over X.*

Proof. We proceed as in Bertsekas and Tsitsiklis [1989]. Let

$$z^i(t) = (x_1(t+1), \ldots, x_{i-1}(t+1), x_i(t+1), x_{i+1}(t), \ldots, x_m(t))$$

From the definition of the iteration (2), we obtain

$$\Phi(x(t)) \geq \Phi(z^1(t)) \geq \Phi(z^2(t)) \geq \ldots \geq \Phi(z^{m-1}(t)) \geq \Phi(x(t+1)) \ \forall t \qquad (4.26)$$

Let $x^* = (x_1^*, \ldots, x_m^*)$ be a limit point of the sequence $\{x(t)\}$. Since X is closed, x^* belongs to X. By the compactness of X, there exists a subsequence $\{x(t_k)\}$ of $\{x(t)\}$ that converges to x^*. Therefore from (3) we observe that $\Phi(x(t_k))$ converges to $\Phi(x^*)$ by the convergence of $\{x(t_k)\}$ to x^* and the continuity of Φ. This implies that the sequence $\Phi(x(t))$ converges to $\Phi(x^*)$. Now we have to show that x^* is a minimizer of Φ over X.

We first show that $z^i(t_k)$ converges to x^*. Let us assume on the contrary or equivalently, that $z^i(t_k) - z^{i-1}(t_k)$ does not converge to zero. Let $\gamma(t_k) = \|z^i(t_k) - z^{i-1}(t_k)\|_2$. By restricting to a subsequence of $\{t_k\}$, we may assume that there exists $\gamma_0 > 0$ such that $\gamma(t_k) \geq \gamma_0$ for all k. Let $s^i(t_k) = (z^i(t_k) - z^{i-1}(t_k))/\gamma(t_k)$. Thus $z^i(t_k) = z^{i-1}(t_k) + \gamma(t_k)s^i(t_k)$, $\|s^i(t_k)\| = 1$, and $s^i(t_k)$ differs from zero only along the ith block component. Notice that $s^i(t_k)$ belongs to a compact set and therefore has a limit point $\bar{s}^i$. By restricting to a subsequence of $\{t_k\}$, we assume that $s^i(t_k)$ converges to $\bar{s}^i$. Let us now fix some $\lambda \in [0,1]$. Since $0 \leq \lambda\gamma_0 \leq \gamma(t_k)$, $z^{i-1}(t_k) + \lambda\gamma(t_k)s^i(t_k)$ lies on the segment joining $z^{i-1}(t_k)$ and $z^{i-1}(t_k) + \gamma(t_k)s^i(t_k)$ and belongs to X because X is convex. Then we have the following

$$\Phi(z^i(t_k)) = \Phi(z^{i-1}(t_k) + \gamma(t_k)s^i(t_k)) \leq \Phi(z^{i-1}(t_k) + \lambda\gamma(t_k)s^i(t_k)) \leq \Phi(z^i(t_k) \qquad (4.27)$$

The first inequality follows from the fact that $z^i(t_k)$ minimizes Φ along the ith block component all the others being held fixed. The second inequality follows from the quasiconvexity of Φ along each block component when the others are held constant.

Since $\Phi(x(t))$ converges to $\Phi(x^*)$, by Eq. (2) $\Phi(z^i(t))$ also converges to $\Phi(x^*)$. When we take the limit as k tends to infinity, we have

$$\Phi(x^*) \leq \Phi(x^* + \lambda\gamma_0\bar{s}^i) \leq \Phi(x^*)$$

We therefore conclude that $\Phi(x^*) = \Phi(x^* + \lambda\gamma_0\bar{s}^i)$, which contradicts the strict quasiconvexity of Φ along the ith block component since $\gamma_0\bar{s}^i \neq 0$. This contradiction establishes that $z^i(t_k)$ converges to x^* for all i.

From the definition (2) of the algorithm, we have for a fixed but arbitrary $x_i \in X_i$

$$\Phi(z^i(t_k)) \leq \Phi(x_1(t_k), x_2(t_k), \ldots, x_i, \ldots, x_m(t_k))$$

When we take the limit as k tends to infinity, we obtain

$$\Phi(x^*) \leq \Phi(x_1^*, x_2^*, \ldots, x_i, \ldots, x_m^*)$$

Therefore by the stationary point condition for constrained optimization, we have

$$\nabla_i\Phi(x^*)'(x_i - x_i^*) \geq 0 \ \ \forall x_i \in X_i$$

Adding these inequalities and by the Cartesian product structure of X, we conclude that

$$\nabla\Phi(x^*)'(x - x^*) \geq 0 \ \ \forall x \in X$$

By the pseudoconvexity of Φ, this shows that x^* minimizes Φ over X. **Q.E.D.**

This result establishes the convergence of the coordinate directions iteration to the solution of the penalty problem. A similar decomposition scheme is proposed in the Ph.D. thesis by Liu [1988] for the minimization of the augmented Lagrangian within the context of block angular optimization. However, no computational experience is reported. Note also that this iteration is not generally parallelizable. As a variant to iteration (4.24) we consider here the iteration

$$x_i(t+1) \ = \ \arg\min_{x_i \in X_i} \Phi(x_{-i}(t))) \tag{4.28}$$

where

$$x_{-i}(t) \stackrel{\text{def}}{=} (x_1(t), \ldots, x_{i-1}(t), x_i, x_{i+1}(t), \ldots, x_m(t))$$

To study the convergence of the iteration (4.28), we will need the following apparatus. Let's consider a mapping $T : X \longmapsto X$ where X is a subset of $\Re^n$ with the property

$$\|T(x) - T(y)\| \leq \alpha\|x - y\| \forall x, y \in X \tag{4.29}$$

Here $\|.\|$ is some norm and α is a constant belonging to $[0, 1)$. Such a mapping is called a **contraction mapping**. We assume that the space $\Re^n$ is represented as the Cartesian product of spaces $\Re_{n_i}$ such that $n = n_1 + n_2 + \ldots + n_m$. We also assume we are given a norm $\|.\|_i$ on $\Re^{n_i}$ for each i and that $\Re^n$ is endowed with the norm

$$\|x\| = \max_i \|x_i\|_i$$

which is called a *block-maximum* norm, Bertsekas and Tsitsiklis [1989]. Let $T : X \longmapsto X$ be a contraction with modulus α, under the above introduced block-maximum norm. Such a mapping is called a *block contraction* in Bertsekas and Tsitsiklis [1989]. Using the above apparatus, we reproduce here without proof the following result.

Proposition 2 *Let $\Phi : \Re^n \longmapsto \Re$ be continuously differentiable, let γ be a positive scalar, and suppose that the mapping $R : X \longmapsto \Re^n$, defined by $R(x) = x - \gamma\nabla\Phi(x)$, is a contraction with respect to the block-maximum norm $\|x\| = \|x_1, \ldots, x_m\| = \max_i \|x_i\|_i/w_i$, where*

each $\|.\|_i$ is the Euclidean norm on $\Re^{n_i}$ and each w_i is a positive scalar. Then, there exists a unique vector x^* which minimizes Φ over X. Furthermore, the nonlinear Jacobi and Gauss-Seidel algorithms are well-defined, that is, a minimizing x_i for iterations (4.24) and (4.28) always exists. Finally the sequence $\{x(t)\}$ generated by either of these algorithms converges to x^* geometrically.

Notice that the above result guarantees convergence of iterations (4.24) as well as iterations (4.28). Although no convexity assumption is made, contraction property is needed for convergence with geometric rate. We note that this result is stronger than Proposition 1 where well-posedness of the Gauss-Seidel scheme was assumed. However, Proposition 2 guarantees well-posedness of both schemes. But it requires the stronger block contraction property.

We now turn into conditions for the mapping $R(x) = x - \gamma\nabla\Phi(x)$ to be a contraction. Let us consider the mapping $T : X \longmapsto \Re^n$ with the ith block component

$$T_i(x) = x_i - \gamma G_i^{-1}\nabla f_i(x) \tag{4.30}$$

where each $f_i : \Re^n \longmapsto \Re^{n_i}$, γ is some scalar and G_i is an invertible symmetric matrix of dimensions $n_i \times n_i$. We assume that X is the Cartesian product of $X_i \subset \Re^{n_i}$. Then we can state the following result.

Proposition 3 *Suppose that X is convex. If $f : \Re^n \longmapsto \Re^n$ is continuously differentiable and there exists a scalar $\alpha \in [0,1)$ such that*

$$\|I - \gamma G_i^{-1}(\nabla_i f_i(x))^t\|_{ii} + \sum_{j\neq i} \|\gamma G_i^{-1}(\nabla_j f_i(x))^t\|_{ij} \leq \alpha \quad \forall x \in X, \forall i \tag{4.31}$$

then the mapping defined $T : X \longmapsto \Re^n$ defined by

$$T_i(x) = x_i - \gamma G_i^{-1}\nabla f_i(x)$$

is a contraction with respect to the block maximum norm $\|.\|$.

Proof. See Bertsekas and Tsitsiklis [1989].

We use $\nabla_j f_i(x)$ to denote the $n_j \times n_i$ matrix whose entries are the partial derivatives of f_i with respect to the components of x_j. If we treat the function f as a gradient mapping, we can specialize this condition to fit in the framework of our Jacobi type iteration.

Let us recall that we denote our objective function Φ for a fixed value of the penalty parameter, then we reassert the contraction mapping condition of Proposition 3 in the following corollary.

Corollary 1 *Suppose that X is convex. If $\Phi : \Re^n \longmapsto \Re$ is twice continuously differentiable and there exists a scalar $\alpha \in [0,1)$ such that*

$$\|I - \gamma \nabla^2_{ii}\Phi(x)^t\|_{ii} + \sum_{j \neq i} \|\gamma \nabla^2_{ji}\Phi(x)^t\|_{ij} \leq \alpha \quad \forall x \in X, \forall i \tag{4.32}$$

then the mapping $R : X \longmapsto \Re^n$ defined by

$$R_i(x) = x_i - \gamma \nabla \Phi_i(x)$$

is a contraction with respect to the block maximum norm $\|.\|$.

We use $\nabla^2_{ij}\Phi(x)$ to denote the $n_i \times n_j$ matrix whose entries are the second derivatives of Φ with respect to the components of x_i and the components of x_j respectively at a given point x.

The results given are obtained under somewhat restrictive assumptions. In particular, the assumptions required for the convergence of the parallel iteration are hard to verify. Furthermore it is not known whether these assumptions are necessary. An empirical testing of these decomposition schemes may reveal some more insight into their properties. Further research in that direction can be pursued in the future for the solution of multicommodity network flows.

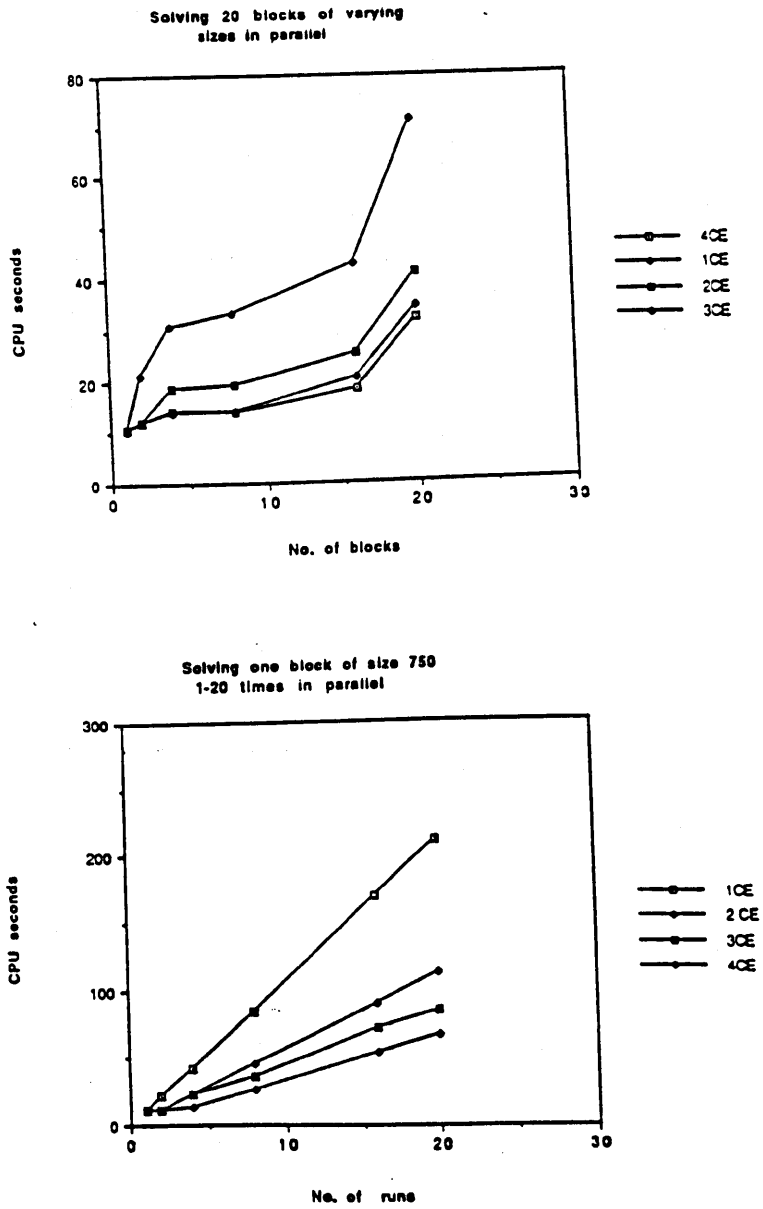


Figure 4.4: Parallel solution of the block–partitioned equations.

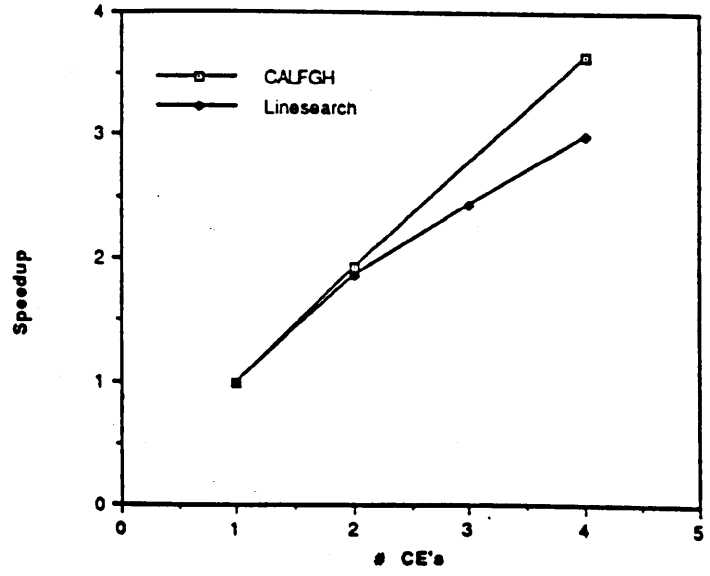


Figure 4.5: Speedup factors of the linesearch.

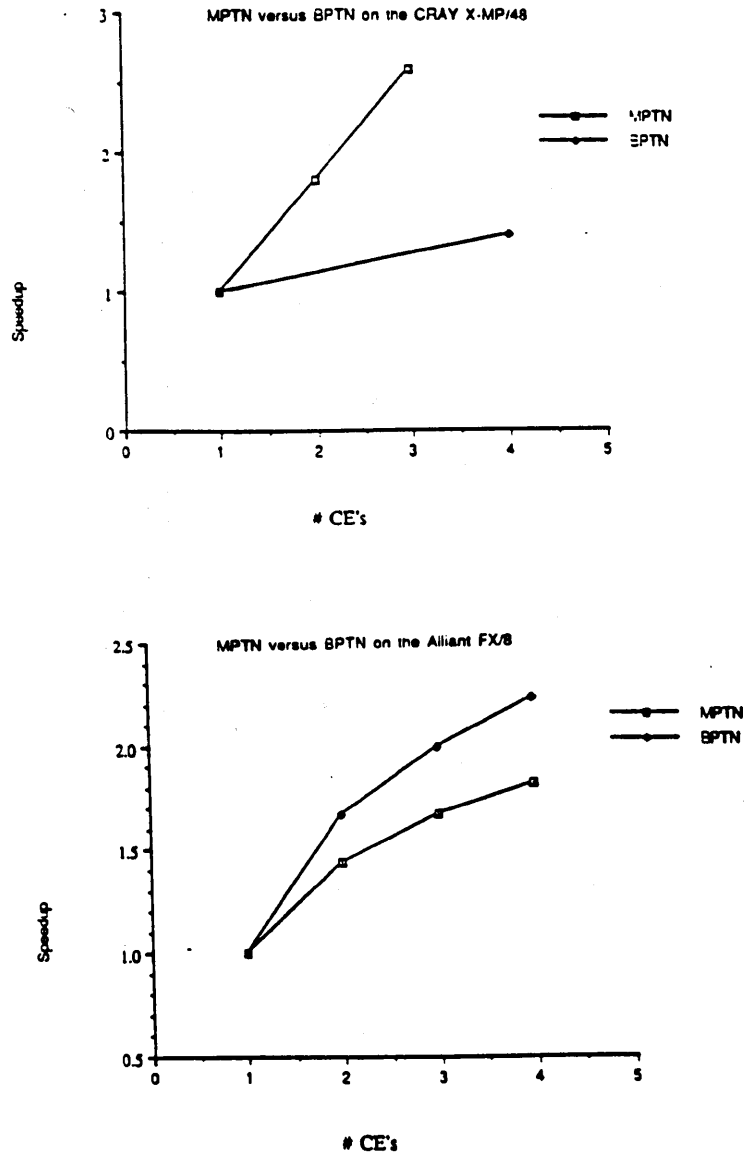


Figure 4.6: Comparison of MPTN and BPTN under different parallel architectures.

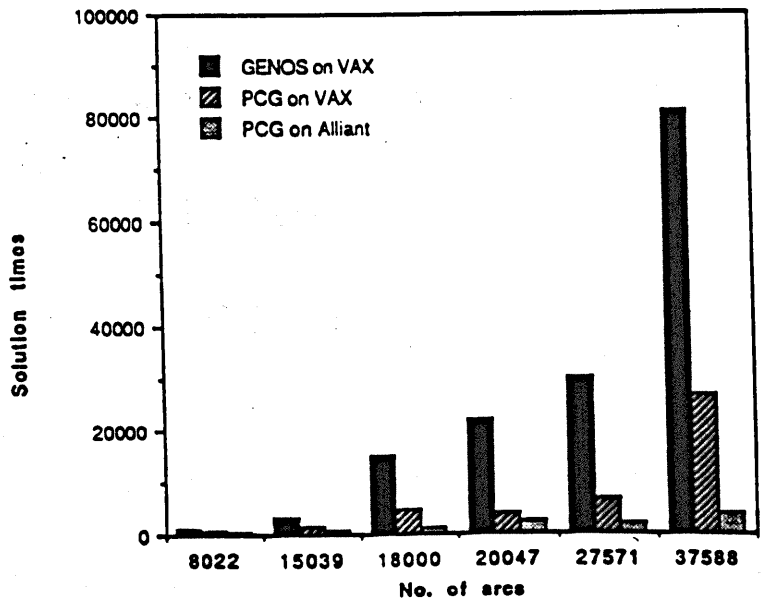


Figure 4.7: Comparative chart of Solution times.

Chapter 5

Coarse Grain Parallel Decomposition of Multicommodity Network Flows

The design of algorithms for the solution of large scale optimization problems can be profoundly influenced by advanced computer architectures. This statement is particularly true for problems with special embedded structures such as the multicommodity network flow problem and the stochastic programming problem with network recourse.

In recent years, vector and parallel supercomputers have been used in numerous computational studies. A survey of the emerging field of parallel optimization can be found in Zenios [1989]. However, most of these efforts were motivated by advances in numerical linear algebra on vector and parallel computers that could naturally be integrated into numerical optimization software. The algorithms studied were inherently sequential but rich in linear algebra computations where vector and parallel computing can offer a significant benefit over a serial computing environment. The philosophy here, however, is substantially different in two important aspects. Vector and parallel computing is exploited not only at the level of linear algebra computations but also at a higher level which induces a decomposition of the problem into subproblems of smaller dimension. First, by ignoring a subset of problem constraints, the problem offers a decomposable structure such as a block angular structure. Furthermore each block may have some exploitable inner struc-

ture which would allow for fast and efficient solution procedures at a lower level. This is the case with multicommodity network flow and stochastic network optimization problems where there is an embedded block angular structure which consists of smaller network flow problems. This decomposition scheme is particularly suitable for *coarse grained* parallel computers. It is therefore this view toward parallelism which dictates the direction to be followed when designing specialized algorithms. Chen and Meyer [1988] exploited this structure for the solution of very large traffic assignment problems. Stochastic programs with network recourse have been analyzed in Mulvey and Vladimirou [1988] much in the same spirit. Zenios, Qi and Armstrong [1991] developed a decomposition algorithm based on the notion of *coercion* functions for the solution of multicommodity network flow problems. A more recent piece of work came from Schultz and Meyer [1991] where they develop a decomposition method based on logarithmic barrier functions coupled with trust regions to solve very large multicommodity network problems. An alternative line of research aims at the design of *massively parallel* decompositions for multicommodity network flows and stochastic programs and see Zenios [1991] and Nielsen and Zenios [1990].

In this chapter, we report on the design and the computational performance of the parallel decomposition algorithm based on linear-quadratic penalty (LQP) functions. Our aim here is to illustrate the parallel decomposition that results from the application of the LQP algorithm and evaluate its performance when implemented on a multiprocessor system.

We proceed with a brief description of some parallel computing concepts and capabilities of the CRAY parallel architectures. The rest of this chapter is organized as follows. In section 5.2 we briefly review the linear–quadratic penalty algorithm. Numerical considerations and parallel decompositions are presented in sections 5.3 and 5.4 along with computational results on a CRAY Y–MP. We conclude in section 5.5 with a critical evaluation of results.

5.1 Overview of Parallel Computing Concepts

Our aim in this section is to give a brief overview of the parallel computing concepts and terminology that will be used throughout this paper. For a more detailed discussion the reader is referred to Zenios and Mulvey [1988].

The computer architecture most suitable for the decomposition framework presented

in the previous section is a vector multiprocessor system. Since the granularity of the job that can be executed concurrently will be large — equal to the number of commodities in the case of multicommodity network flow problem — the use of a coarse grained parallel architecture with a small number of powerful processors is well justified. The CRAY Y-MP used in this study is equipped with eight vector processors. Upon compilation, the CRAY Y-MP vectorizing FORTRAN compiler generates machine code that makes efficient use of the hardware features. However, the user may find it beneficial to rewrite part of a program to make more efficient use of the vector features of the system. Further improvements in performance can be achieved through the use of BLAS routines for linear algebra computations contained in SCILIB subroutine library. SCILIB is a library that offers higly optimized versions of common scientific mathematical routines. SCILIB contains subroutines which perform inner products(vector-vector), products of a matrix and a vector (matrix-vector) and products of two matrices (matrix-matrix).

Two modes of parallel computing are considered in this study: vectorization and multitasking. We review each concept briefly. Vectorization can be defined as the restructuring of a program in order to exploit parallelism at the innermost DO-loop level. Vector processing allows operations on long arrays to be carried out in roughly the same amount of time required for scalar computations. However, iterations of a loop should be computationally independent to ensure efficient vectorization and correctness of results. Vectorization is automated by means of vectorizing compilers as in the case of CRAY Y-MP. Nevertheless some restructuring of DO-loops with a view towards efficient vectorization may be necessary to eliminate dependencies that could inhibit vectorization. A description of CRAY vectorization features and recent enhancements can be found in CRAY [1989].

Multitasking is defined as the structuring of a program into two or more components that can execute concurrently on multiple processors. The units of computation that are candidates for concurrent execution are called *tasks*. One copy of a program module is used by multiple tasks for parallel execution. This property of a program is called *reentrancy*. The module is typically a subroutine or a set of program statements. In addition to reentrancy, independence of parallel tasks is required to ensure correct execution of a concurrent program. Parallel tasks of a concurrent program should be computation and storage independent. A more detailed description of independence concepts can be found in CRAY [1987].

The multitasking features on the CRAY Y-MP come in two options: macrotasking and microtasking. Macrotasking is typically implemented at the subroutine level. Modules that can execute in parallel are coded as subroutines that are activated for macrotasking through calls to library routines. On the other hand, microtasking allows the programmer to define tasks within a program unit by compiler directives. The sections of a code that can be microtasked are typically the iterations of an outermost loop(that may contain subroutine calls) or iterations of an innermost loop. Microtasking is achieved with the use of a set of preprocessor directives. These directives delimit segments of code that are independent and provide suitable synchronization and locking mechanisms for shared data which can be accessed by all concurrently executing processors. The use of microtasking on the CRAY supercomputers have been further facilitated by the Autotasking features which operate with a set of simpler directives. Autotasking directives are interpreted by an intermediate preprocessor which may modify the code to make better use of microtasking and insert microtasking directives. The user may bypass autotasking by analyzing the code and inserting microtasking directives. But the use of autotasking is preferable since it considerably simplifies the programmer's work.

5.2 The Linear-Quadratic Penalty Method for Multicommodity Network Flows

In this section we briefly review the main ingredients of the linear-quadratic penalty algorithm discussed in Chapter 4. We solve the penalty problem using simplicial decomposition. We briefly describe the main ingredients of simplicial decomposition algorithm here to pave the way for the forthcoming sections on vector and parallel computations.

5.2.1 Overview of Simplicial Decomposition Computations

Recall that simplicial decomposition iterates by solving a sequence of linear problems to generate vertices of the polytope X. A nonlinear master problem optimizes the penalized objective function $\Phi_{\mu,\epsilon}$ on the simplex specified by the vertices generated by the subproblems.

At Step 1 the algorithm solves a linear approximation to the nonlinear program. If the set X has a Cartesian product structure the problem can be solved independently for each

block of the constraint matrix A:

Step 1 (Decomposed linear subproblems)

For each $\ell = 1, 2, \ldots, K$ solve

$$\begin{aligned} \text{Minimize} \quad & y_\ell^T \nabla_\ell \Phi_{\mu,\epsilon}(z^\nu) \\ \text{subject to} \quad & \\ & A_\ell y_\ell = b_\ell \\ & 0 \leq y_\ell \leq u_\ell \end{aligned}$$

We use ℓ to index the ℓ-th block of the constraint matrix $A = \text{diag}[A_1, A_2, \ldots, A_K]$ and the ℓ-th block of the gradient vector and problem variables. The subproblems are linear min-cost network flow problems and can be solved very efficiently with the network simplex method.

The master program can be written in the form:

$$\begin{aligned} \underset{w}{\text{Minimize}} \quad & \Phi_{\mu,\epsilon}(Bw) \\ \text{subject to} \quad & \\ & \sum_{i=1}^{v} w_i = 1 \\ & w_i \geq 0 \quad i = 1, \ldots, v \end{aligned}$$

where $B = [y^1 | y^2 | \ldots | y^v]$ is the basis for the simplex generated by the vertices $y_1, y_2, \ldots, y_v$. By elimination of the simplex constraint we obtain the equivalent nonlinear program:

$$\min_{w \geq 0} \Phi_{\mu,\epsilon}(Dw) \tag{5.1}$$

where $D = [y_1 - y_v | y_2 - y_v | \ldots | y_{v-1} - y_v]$ is the derived linear basis for the simplex generated by the vertices $y_1, y_2, \ldots, y_v$. We denote by w the vector $[w_1, w_2, \ldots, w_{v-1}]$ and the solution for w_v is computed as

$$w_v = 1 - \sum_{i=1}^{v-1} w_i \tag{5.2}$$

If the vertices that carry zero weight are dropped, the problem becomes a locally unconstrained nonlinear program: recall that at the current iteration we have $v-1$ active vertices (i.e. $w_i > 0$, for $i = 1, \ldots, v-1$) and the last vertex y_v lies along a direction of descent. Hence, given a point $z^\nu \in X$ a descent direction to (5.1) can be obtained as the solution to

$$(D^T M D)p = -D^T \nabla \Phi_{\mu,\epsilon}(z^\nu), \tag{5.3}$$

where the choice of the matrix M is discussed in detail in Chapter 4 of this manuscript. Practical experience shows that M should be taken to be an approximation to the second derivative matrix of the function $\Phi_{\mu,\epsilon}$ to achieve the best performance with the algorithm. We define the matrix $C = (D^T M D)$ for ease of notation. Having computed a descent direction in the space of simplicial weights, the algorithm proceeds with a one dimensional search procedure to minimize the penalized objective function $\Phi_{\mu,\epsilon}$ along this direction. A concise description of the master problem algorithm can be given as follows:

Master Problem Algorithm

Do until some convergence criteria are satisfied

1 Compute

$$C = D^T M D$$

2 Compute search direction p as the solution to

$$Cp = -D^T \nabla \Phi_{\mu,\epsilon}(z^\nu),$$

3 Compute maximum step w_d along this direction

4 One dimensional search along the direction $p_d = w_d - w$ solve for α^*

$$\alpha^* = \arg \min_{0 \le \alpha \le 1} \Phi_{\mu,\epsilon}(D \cdot (w + \alpha p_d) + y_v)$$

5 Move

$$w \leftarrow w + \alpha^* p_d$$

End do.

Details can be found in Von Hohenbalken [1977] and Mulvey, Zenios and Ahlfeld [1990]. The master problem algorithm summarized above involves dense linear algebra computations. Practical experience shows that as the problem size gets larger, these computations tend to dominate the total solution time. We focus on these computations in the next section.

5.3 Vector and Parallel Computations with the Master Problem

One of the primary motivations for the choice of simplicial decomposition — besides the separability offered by linearization — lies in the structure of the master problem. The master problem algorithm described in section 5.2.1 is very rich in dense linear algebra computations that are suitable for both vector and parallel computations. In this section we are concerned with the impact of vector and parallel computing on the performance of the algorithm. We identify the computation intensive modules in the master problem algorithm and describe efficient ways to modify the code for vector and parallel execution in the following sections.

5.3.1 Computing Descent Directions

The master problem algorithm computes a descent direction obtained as the solution of the system

$$Cp = -D^T \nabla \Phi_{\mu,\epsilon}(z^\nu),$$

As practical experience shows the computation of the matrix C dominates the master problem computations and constitutes a large fraction of the total execution time of the LQP algorithm. Although the matrix C is only $(v-1) \times (v-1)$, its computation involves a second derivative matrix H of dimensions $Kn \times Kn$ and a matrix D of dimensions $Kn \times (v-1)$. The matrix H is usually very sparse but the matrix D is completely dense. The matrix C is computed in the form:

$$C = B^T H B - B^T H \hat{Y} - \hat{Y}^T H B + \hat{Y}^T H \hat{Y} \tag{5.4}$$

where $\hat{Y}$ is a conformable matrix whose columns are identical and have value y_v, and $B = [y^1|y^2|\ldots|y^v]$ is the basis for the simplex generated by the vertices $y_1, y_2, \ldots, y_v$. Computation of C can be accomplished in two phases. In the first phase, the following computation is performed

$$B^T H B - \hat{Y}^T H B.$$

Since C is a symmetric matrix, it is stored as an upper triangular matrix in column format, see Figure 5.1, and the product $B^T H B$ can be performed in such a way as to form only

the upper triangular part of C. We assume that we have v vertices in the matrix B and we use B_i to denote the i-th column of the matrix B. The first phase computations can be compactly described as follows:

for all $i = 1, \ldots, v$ do
    compute $p = HB_i$
        $q = y_v^T p$
    for all $j = 1, i$ do
        $r = B_j^T p$
        $k = j + i(i-1)/2$
        $c(k) = r - q$
    end for
end for

It can be easily verified that the above procedure operates on entries of column i of the matrix C at each iteration i of the outer loop. In the second phase, the expression

$$\hat{Y}^T H \hat{Y} - B^T H \hat{Y}$$

is computed and added to C. We describe the procedure as follows:

Compute $p = HB_v$
    $q = B_v^T p$
    for all $i = 1, \ldots, v-1$ do
        compute $r = p^T B_i$
        for all $j = 1, \ldots, v-1$ do
            $k = i + j(j-1)/2$
            $c(k) = c(k) + q - r$
        end for
    end for

Again it can be verified that at each iteration i of the outer loop, only the entries of row i of C are updated. The inner product computations can be performed using the SCILIB

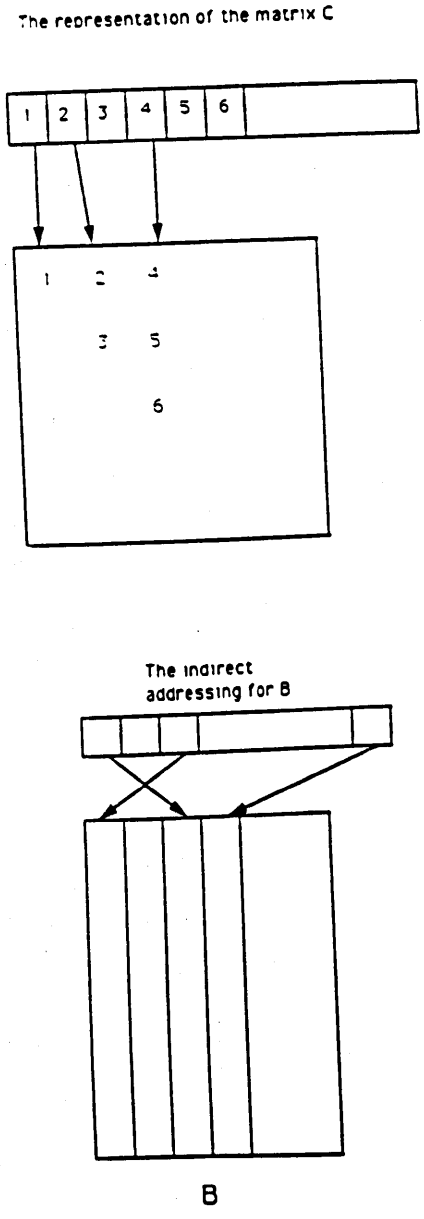


Figure 5.1: Upper triangular representation of the matrix C and the indirect addressing scheme for the matrix B.

routine **SDOT**. Furthermore, by the independence of loop iterations, both procedures can execute on multiple processors concurrently. This can be achieved by making the inner loop computations a separate subroutine in each phase. The modifications performed on the FORTRAN code are given in Appendix B. The impact of autotasking the modified code is discussed in section 5.3.4.

An alternative way to exploit parallelism in the computation of C is through the use of matrix-matrix and matrix-vector SCILIB routines. In the implementation of simplicial decomposition, a pointer addressing scheme is used to indicate the active vertices stored in the matrix B. This is necessary to avoid rearranging the columns of the matrix B every time a vertex is added or dropped. The indirect addressing is also depicted in Figure 5.1. This scheme prohibits the use of matrix-matrix and matrix-vector SCILIB routines **MXM** and **MXV**. The indirect addressing scheme was removed to enable the use of these routines in the computation of the matrix C. The columns of the matrix B need to be rearranged in the absence of the indirect addressing scheme. The use of **MXM** and **MXV** routines produced only a marginal improvement in performance over the use of **SDOT** routine for the computation of inner products. All computational tests reported in this study have been conducted using the indirect addressing.

5.3.2 Function and Gradient Evaluations

Having computed a descent direction, a one-dimensional search is executed to compute the next iterate. The time spent in the search procedure is dominated by the computation of function values and the gradient vector. The function and gradient evaluation of the original objective function f can be vectorized trivially as it involves a simple DO-loop over all variables in the problem. However, the function and gradient values contributed by the nonseparable penalty function requires the evaluation of the side constraints $y_j = (Ex-d)_j$ for all $j = 1,2,\ldots,s$. If side constraint j is satisfied by the current iterate x, the penalty function and derivative values vanish since y_j is less than or equal to zero. Otherwise the penalty term is either the quadratic or the linear term as given by

$$\phi(\epsilon, y_j) = \begin{cases} 0 & \text{if } y_j \leq 0 \\ \frac{y_j^2}{2\epsilon} & \text{if } 0 \leq y_j \leq \epsilon \\ y_j - \frac{\epsilon}{2} & \text{if } y_j \geq \epsilon \end{cases}$$

This procedure can be vectorized in two phases. In the first phase, the degree of violation is computed and the appropriate penalty term is identified for each side constraint. Then in the second phase, the function and gradient values are computed in a simple DO-loop based on the information obtained from the first phase. The procedure can be described as follows:

For all arcs $j = 1, \ldots, n$ do
    compute $y_j = (Ex - d)_j$
    if $(y_j \leq 0)$ no penalty
    else if $(y_j < \epsilon)$ quadratic segment
        else linear segment
end for

For all arcs in quadratic segment do
    evaluate penalty function and derivative
end for

For all arcs in linear segment do
    evaluate penalty function and derivative
end for

This procedure eliminates any conditional branching scheme which could inhibit vectorization by splitting the computation into two distinct phases.

5.3.3 Other Linear Algebra Computations

Computation of the Reduced Gradient

The solution of the system

$$Cp = -D^T \nabla \Phi_{\mu,\epsilon}(z^\nu),$$

at every step of the master problem also requires the computation of the right-hand side reduced gradient vector $D^T \nabla \Phi_{\mu,\epsilon}(z^\nu)$. This is a fully dense matrix-vector product that

can be efficiently vectorized. However, the matrix D is never explicitly formed as we already mentioned in section 5.3.1 and the indirect addressing scheme used to access columns of the matrix B reduces the efficiency of the vectorization somewhat. By removing the indirect addressing, the matrix-vector product routine **MXV** can be used in this calculation. Alternatively, this matrix-vector product can be parallelized in the same spirit as in the computation of the matrix C. However this direction was not pursued since the computation time spent in this module was already sufficiently reduced through compiler vectorization.

Evaluation of Side Constraints

During the linesearch procedure, the computation of the penalty function requires the evaluation of the side constraints at the current iterate. In the case of multicommodity flows, the side constraints are the mutual arc capacity constraints on some or all the arcs of the network. Hence, the total flow can be computed in a straightforward way by adding flow variables for each commodity. This computation has to be performed before each function evaluation and can be vectorized automatically by the compiler. Alternatively, this computation can be interpreted as a dense SAXPY (i.e. $Y = Y + aX$) operation. The Linear Algebra Library routine **SAXPY** is used in this procedure.

5.3.4 Numerical Results

In this section we report on the impact of vector and parallel computing with the master problem on the overall performance of the LQP algorithm. We will illustrate the improvement in performance produced as a result of exploiting vector and parallel computing with the master problem algorithm on a set of large multicommodity network flow problems. We reproduce the test problem characteristics here.

Test problem	No. of arcs	No. of nodes	No. of rows	No. of columns
PDS1	339	126	1473	3816
PDS3	1117	390	4593	12590
PDS5	2149	686	7546	23639
PDS10	4433	1399	15389	48763
PDS15	7203	2125	23375	79233
PDS20	10116	2447	31427	105728
PDS30	15526	4223	46453	154998

Table 5.1: Test Problem Characteristics

Vector Performance of the Algorithm

In this section we present computational results with the scalar and vectorized version of the linear–quadratic penalty algorithm. Before we proceed with vector computing, we give in Table 5.2 the computational results with three PDS problems on a VAX 6400 mainframe computer. The solution time has two main components: the time to solve the network subproblems and the time to solve the master problems. All times are stated in CPU or elapsed seconds unless otherwise indicated. The VAX 6400 is running VMS 5.3 and and the CRAY Y-MP is running UNICOS 5.10. All computational tests were performed using an identical control parameter set for each problem.

Test Problem	Subproblem time	Master problem time	Total time
PDS1	7.5	104.8	112.3
PDS3	63.09	943.59	1006.68
PDS5	324.33	6641.33	6966.66

Table 5.2: Performance of the linear-quadratic penalty algorithm on the VAX 6400.

In Table 5.3 we illustrate the performance of the algorithm on a CRAY Y–MP with vector computing.

Test Problem	Subproblem time	Master problem time	Total time
PDS1	1.01	0.85	1.86
PDS3	12.04	6.73	18.77
PDS5	55.12	37.97	93.10

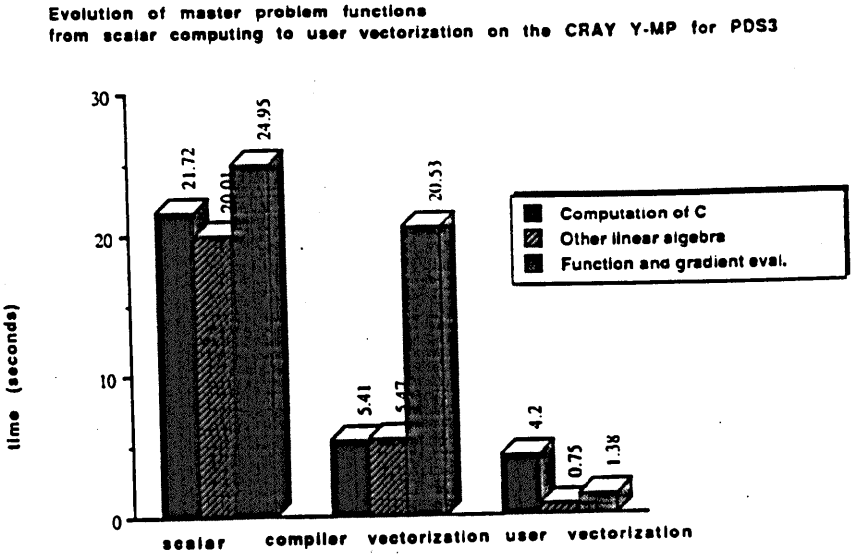
Table 5.3: Performance of the vectorized linear-quadratic penalty algorithm on the CRAY Y-MP.

Test Problem	Subproblem time	Master problem time	Total time
PDS10	232.31	175.82	408.11
PDS15	559.35	381.08	940.43
PDS20	1225.83	720.06	1945.86
PDS30	5536.61	1967.38	7504

Table 5.4: Solution times on the CRAY Y-MP for large problems with single processor vectorization

To illustrate the gains in performance in a vector processing environment, we give in Figure 5.2 an evolution of master problem solution time through a breakdown into its main functions: computation of the matrix C, function and gradient evaluations (one dimensional search), other linear algebra (computation of reduced gradient, evaluation of side constraints). The figure is based on results obtained on the CRAY Y-MP with **1**.scalar computing (no vectorization) **2**.compiler vectorization **3**.user vectorization (through restructuring of the FORTRAN code and use of SCILIB routines as discussed throughout section 5.3). The difference between the total master problem solution time and the total time spent in the main master problem functions accounts for various initializations and other computations such as the evaluation of the second derivative matrix. As evidenced by the results, significant gains are realized in master problem functions with user vectorization.

We provide in Figure 5.3 the change in subproblem and master problem solution times when moving the algorithm from a serial computer to a vector computing environment. A breakdown of the total solution time between subproblem and master problem solution



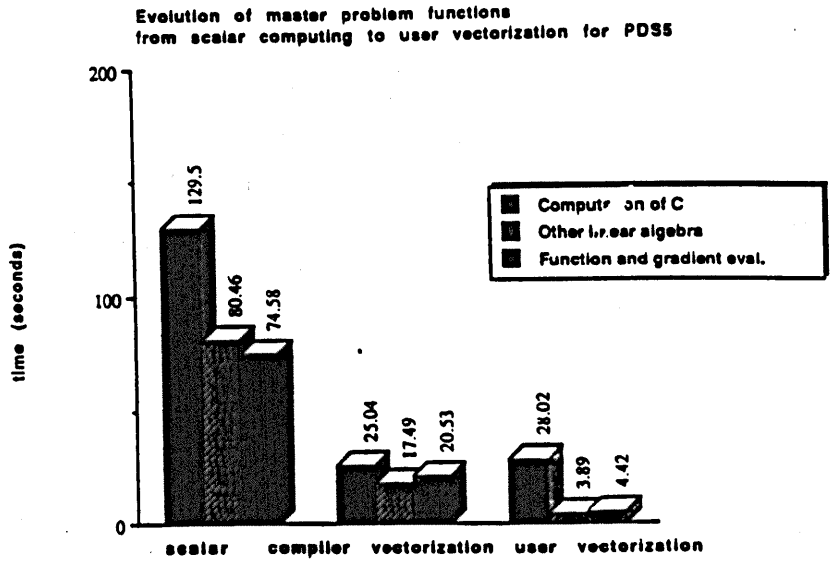


Figure 5.2: Evolution of the master problem functions on the CRAY Y-MP with 1.scalar computing (no vectorization) 2.compiler vectorization 3.user vectorization (through restructuring of the FORTRAN code and use of SCILIB routines as discussed throughout section 5.3).

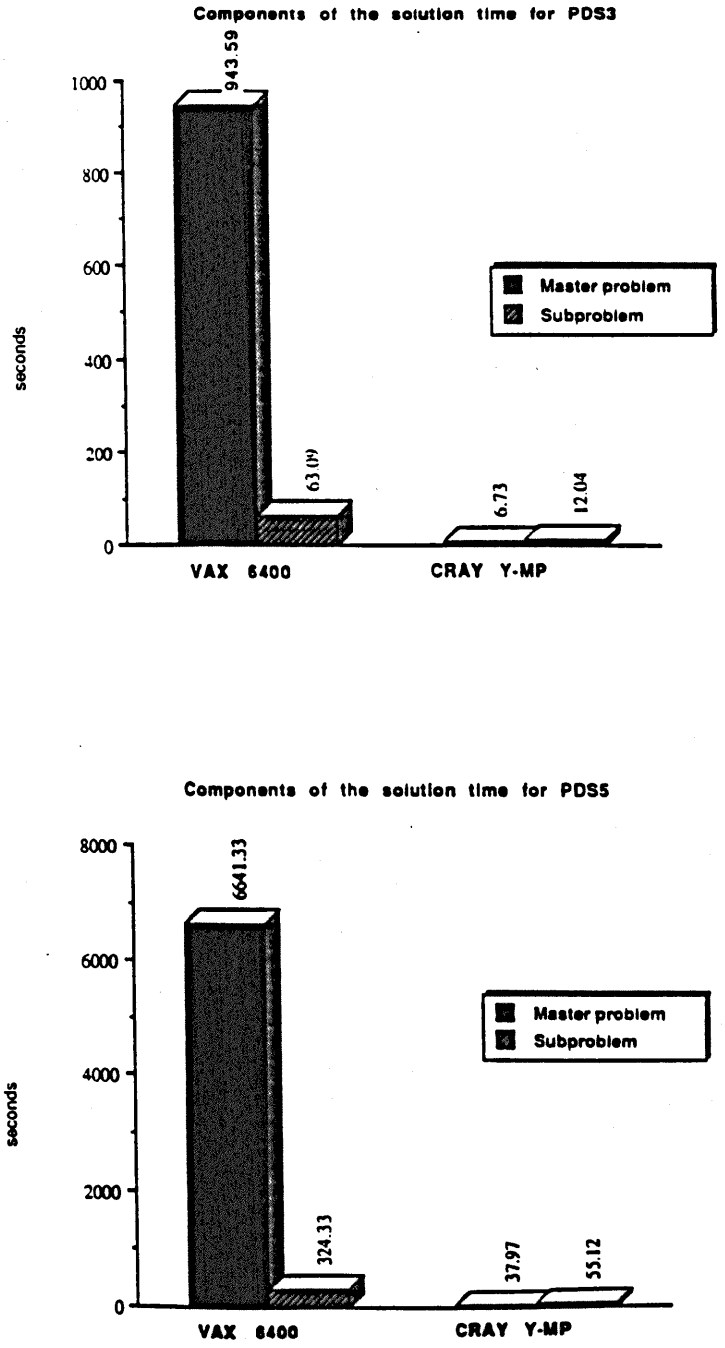
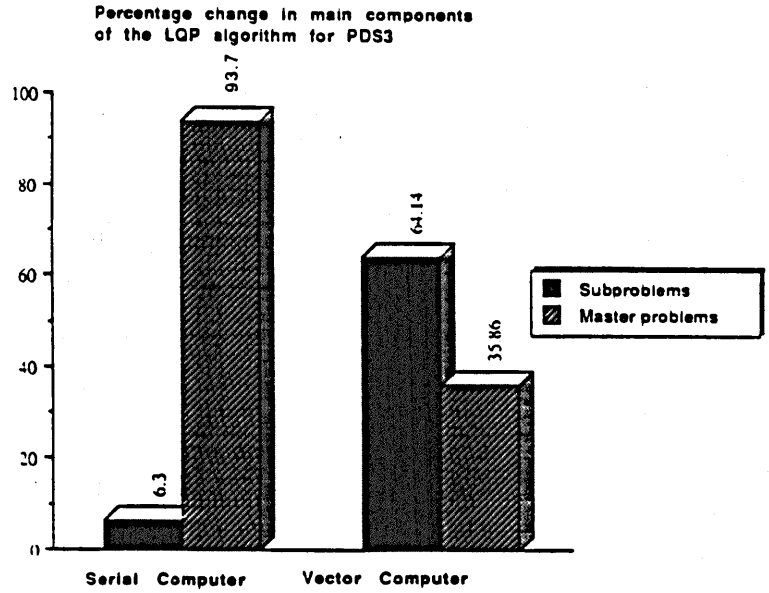


Figure 5.3: Components of the total solution time for the LQP algorithm on serial (VAX 6400) and vector (CRAY Y-MP) computers.

indicates that while the master problem algorithm dominates the computation time in a serial computing environment, the two components are at par after the code is vectorized on the CRAY Y-MP with the code restructuring and SCILIB routines. This phenomenon is further illustrated in Figure 5.4 where we contrast the percentage share of time consumed in the subproblem phase and master problem phase in serial and vector computing environments.

With larger problems, the subproblem time starts to dominate the computation time by a significantly large margin. This is illustrated in Table 5.4 where we give the computational results with larger PDS problems. It is not possible to improve significantly the subproblem solution time through vectorization due to the inherently combinatorial nature of the network simplex algorithm. Parallel decomposition of the subproblem phase will be discussed in section 5.4.



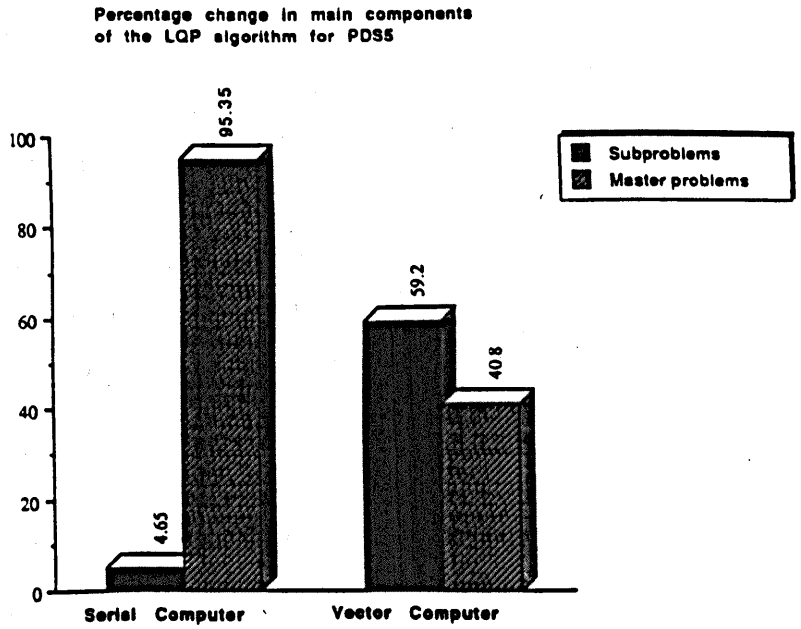


Figure 5.4: Percentage distribution of the total solution time for the LQP algorithm on serial (VAX 6400) and vector (CRAY Y-MP) computers.

Parallel Performance of the Master Problem Algorithm

As we argued throughout section 5.3 it is possible to exploit parallelism at the master problem through the autotasking features of the CRAY Y-MP. Each phase of the two-phase procedure introduced in section 5.3.1 were executed on multiple processors. The effects of autotasking the computation of the product

$$C = B^T H B - B^T H \hat{Y} - \hat{Y}^T H B + \hat{Y}^T H \hat{Y}$$

is illustrated in Table 5.5 on a set of four test problems. The solution times are given in elapsed wall clock time.

Test Problem	Subproblem time	Master problem time	Total time
PDS3	12.03	3.45	15.48
PDS5	55.75	15.37	72.11
PDS10	234.45	60.93	295.38
PDS15	559.	133.	692.

Table 5.5: Solution times on the CRAY Y-MP on 8 processors with autotasked linear algebra routines.

In Figure 5.5, we illustrate the performance of the algorithm using Amdahl's law. We use the following version of Amdahl's law:

$$\frac{T_1}{T_p} = \frac{1}{a/p + (1-a)}$$

where

T_1 is the single processor execution time

T_p is the execution time on p processors.

a is the fraction of the code that can be executed in parallel.

The fraction a is taken to be the ratio of the cumulative time spent in the computation of the matrix C to the total solution time. This computation may account for more than 80% of the master problem time for large problems and approximately 30% of the total solution

time. As can be observed from the figure, although there is only a modest improvement in the total solution time the parallel performance of the module responsible for the computation of C produces an overall speedup in the algorithm which is very close to the upper bound given by Amdahl's law.

We conclude that significant gains in performance can be achieved by taking advantage of the vector capabilities of a particular parallel architecture. Further enhancements are possible with the use of SCILIB linear algebra routines and some restructuring of the FORTRAN code. There is also a high payoff in exploiting parallelism using multiple processors through a detailed analysis of computations and a proper modification of the FORTRAN code.

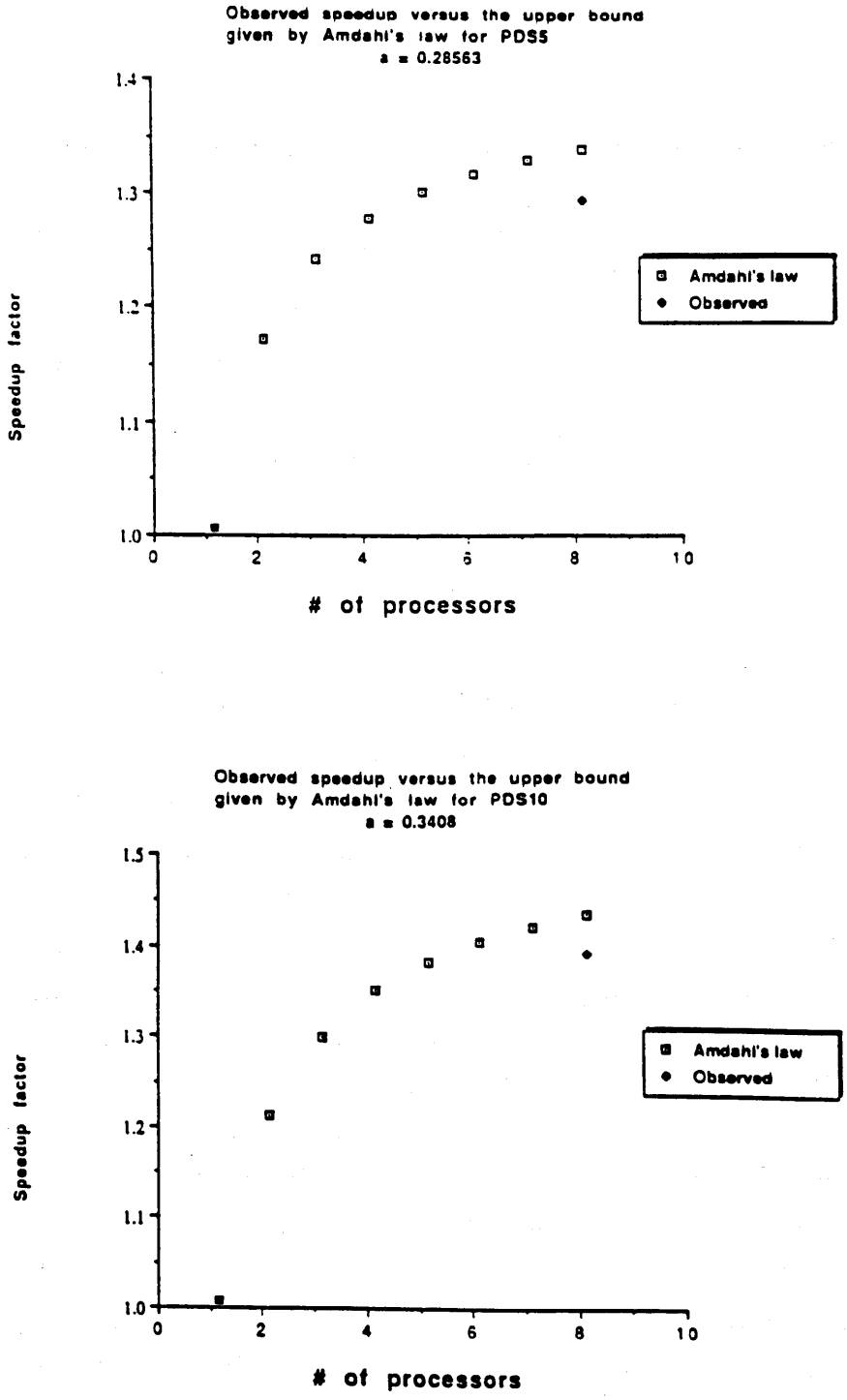


Figure 5.5: Performance of the algorithm with parallelized master problem computations with respect to Amdahl's law

5.4 Parallel Decomposition of Network Subproblems

As we discussed in section 3.1, simplicial decomposition allows the subproblems to be decoupled into smaller network flow problems. During the subproblem phase, the algorithm solves K linear programs:
For each $\ell = 1, 2, \ldots, K$ solve

$$\begin{aligned} \text{Minimize} \quad & y_\ell^T \nabla_\ell \Phi_{\mu,\epsilon}(z^\nu) \\ \text{subject to} \quad & \\ & A_\ell y_\ell = b_\ell \\ & 0 \leq y_\ell \leq u_\ell \end{aligned}$$

The linear min-cost network flow problem for each commodity can be solved using the network simplex method. The linear network optimizer LPNETG of Mulvey and Zenios [1987] is used to solve the subproblems. LPNETG can be called as a subroutine from the simplicial decomposition code. To solve the network problem for each commodity, LPNETG is called from within a DO-loop. At each iteration k of the loop, data relevant to commodity k is passed to the network solver. The DO-loop was autotasked using the appropriate compiler directive which allows concurrent subroutine calls. Thus, LPNETG is made *reentrant*. However, LPNETG uses some global scalar variables stored in COMMON blocks. Global variables require special attention since they are accessible to all processors. To ensure correctness of results, global variables need to be made private to each processor for the duration of the parallel execution phase. This is accomplished through the use of CRAY FORTRAN extension TASK COMMON which makes global data local to all concurrently executing processors. During parallel execution of LPNETG for each commodity, each processor has access to a temporary local copy of the TASK COMMON variables. TASK COMMON blocks are interpreted as regular COMMON for code segments that are executed by a single processor.

We summarize in Table 5.6 the performance of the parallel decomposition algorithm on eight processors. Results are given in elapsed wall clock time. To further illustrate the efficiency of the autotasked code, we present in Figure 5.6 a comparison of the speedup estimate given by Amdahl's law to the observed speedup $\frac{T_1}{T_p}$ where T_p is the execution time

on p processors.

Test Problem	Subproblem time	Master problem time	Total time
PDS3	1.32	3.15	4.47
PDS5	6.96	16.00	22.96
PDS10	33.16	62.86	96.02
PDS15	174.51	133.22	307.33

Table 5.6: Performance of parallel decomposition on the CRAY Y-MP on 8 processors. The performance of the parallel decomposition with larger problems is given in Table 5.7.

Test Problem	Total solution time
PDS20	740
PDS30	2566

Table 5.7: Performance of parallel decomposition on the CRAY Y-MP on 8 processors with larger problems.

Amdahl's law assumes that all processors have identical work load. However, the network subproblems that are solved in parallel may not be of equal degree of difficulty. This is indeed the case with the PDS problems used in this study. For some commodities the number of network simplex pivots to optimality are very large compared to others. This situation is illustrated in Figure 5.7 for PDS20 using the results of an arbitrary subproblem phase. This is a typical load balancing problem which can help explain the gap between the upper bound provided by Amdahl's law and actually observed speedup. To obtain a tighter upper bound on the speedup factor, we can make use of the information contained in Figure 5.7. Let p_k be the number of network simplex iterations for commodity k. Assuming each iteration takes unit time, the total sequential computation time T_1 for an arbitrary subproblem phase is given by

$$T_1 = \sum_{k=1}^{K} p_k$$

Using the data in Figure 5.7, T_1 is computed to be 24054 units. Ideally, the expected

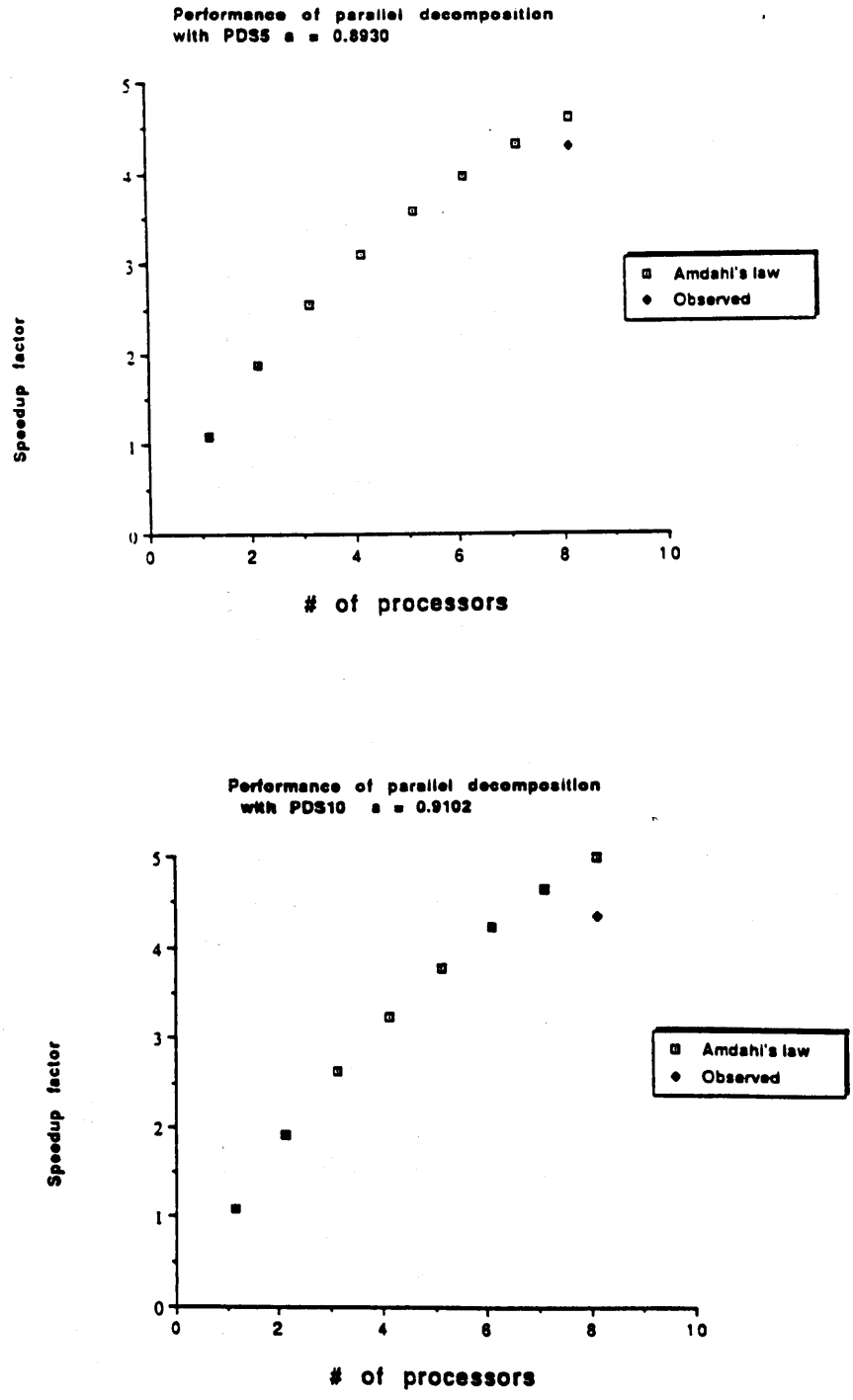


Figure 5.6: Performance of the parallel decomposition with respect to Amdahl's law

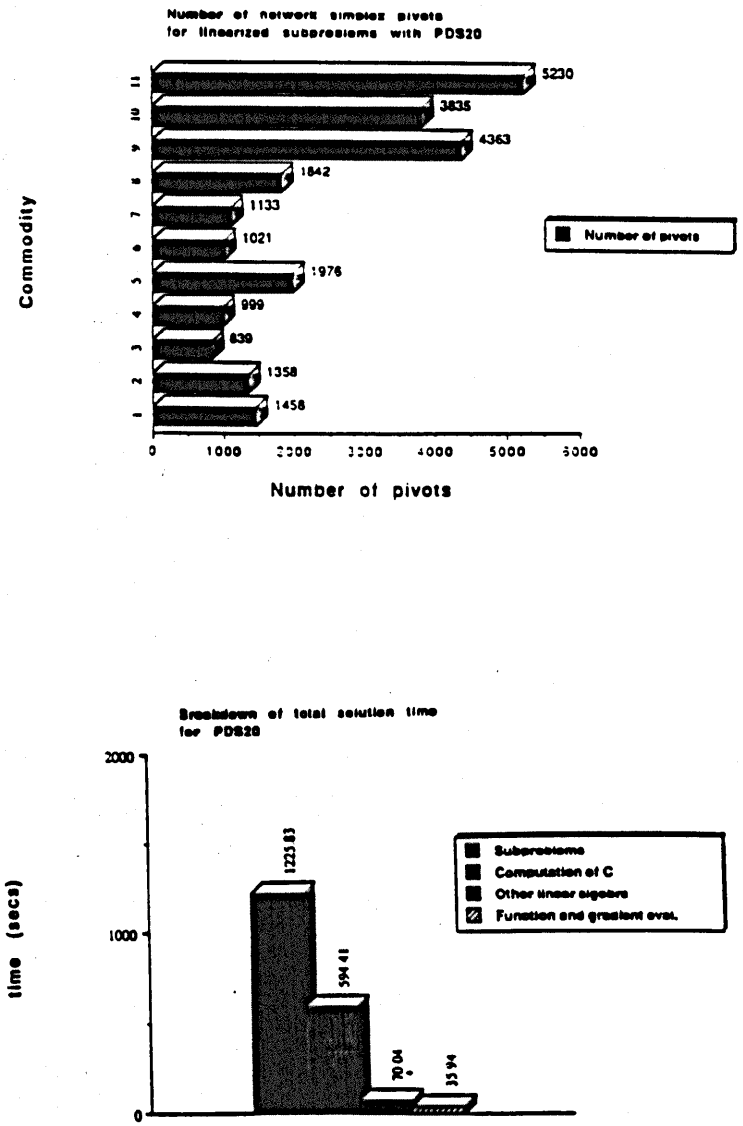


Figure 5.7: Number of network simplex pivots for each commodity with PDS20 and components of the solution time

completion time on p processors would be

$$T_p = \frac{T_1}{p}$$

However, since the values of p_k differ significantly an estimate $\tilde{T}_p$ by assuming that the p processors will pick up the first p commodities first. Applying this principle to the data of Figure 5.7, we get $\tilde{T}_8$ on eight processors to be 6251 units. Now let a_1 be the fraction of total execution time from the master problem that can be executed concurrently and a_2 the fraction from the subproblem phase. In direct application of Amdahl's law, we would take $a_1 = 0.3054$ and $a_2 = 0.6299$ for PDS20 (A breakdown of the total solution time into the subproblem and main master problem functions for PDS20 is given in Figure 5.7) and use the formula

$$s_p \equiv \frac{T_1}{T_p} = \frac{1}{\frac{(a_1+a_2)}{p} + (1 - a_1 - a_2)}$$

The expected speedup s_8 on eight is thus computed to be 5.50. However, using a modified estimate $\tilde{a}_{2p} = \frac{\tilde{T}_p}{T_1} = 0.2598$, we can use the formula

$$\tilde{s}_p \equiv \frac{T_1}{T_p} = \frac{1}{(\frac{a_1}{p} + \tilde{a}_{2p}) + (1 - a_1 - a_2)}$$

The upper bound $\tilde{s}_8$ thus obtained is 2.757 compared to the actually observed speedup of 2.628 on eight processors for PDS20. There are, however, two assumptions inherent in the above modification. First, it is assumed that the relative iteration number pattern holds throughout execution of the algorithm. This seems to be a reasonable assumption supported by computational evidence. The second assumption is the processing order rule. Nevertheless, the analysis yields a more realistic bound on the performance of the parallel decomposition that can be expected under load imbalances.

Another limitation is the number of processors available on the system which does not match the number of tasks scheduled for parallel execution (i.e. the number of commodities). A possible remedy to this problem would be to solve the network subproblems for bottleneck commodities only approximately thereby reducing the number of network simplex iterations. Alternatively commodities which take a relatively small number of pivots can be merged into groups prior to parallel execution therefore reducing the granularity of the tasks to achieve a better load balance.

We illustrate the evolution of the solution time from the vectorized version of the algorithm to parallel decomposition in Figure 5.8. A factor of improvement between 3 to 5 is observed for all problems as a combined effect of vectorization and parallel decompositions.

Finally we provide in Figure 5.9 a comparison of the parallel LQP method with the interior point code OB1 on the CRAY Y-MP. Results with OB1 were obtained on a single processor with vectorization. As is apparent from the figure, the vectorized version of the LQP algorithm already outperforms the original implementation of primal-dual interior point methods on PDS10 and PDS20 significantly. However, this is only a partial comparison since OB1 delivers more accurate solutions than the LQP algorithm.

5.5 Conclusions

We have developed in this chapter alternative decomposition schemes for large scale problems. The algorithm is based on parallel decompositions at the subproblem level into independent network problems and a nonlinear master problem where vectorization and parallelism can be exploited with the linear algebra computations. The numerical results obtained with a set of large multicommodity network problems give strong empirical evidence that it is a worthwhile research effort to design specialized algorithms with a high parallelism potential. The algorithm also outperforms earlier implementations of interior point methods even without using parallel processors. However, the load imbalance remains a problem which plagues the parallel decomposition of subproblems.

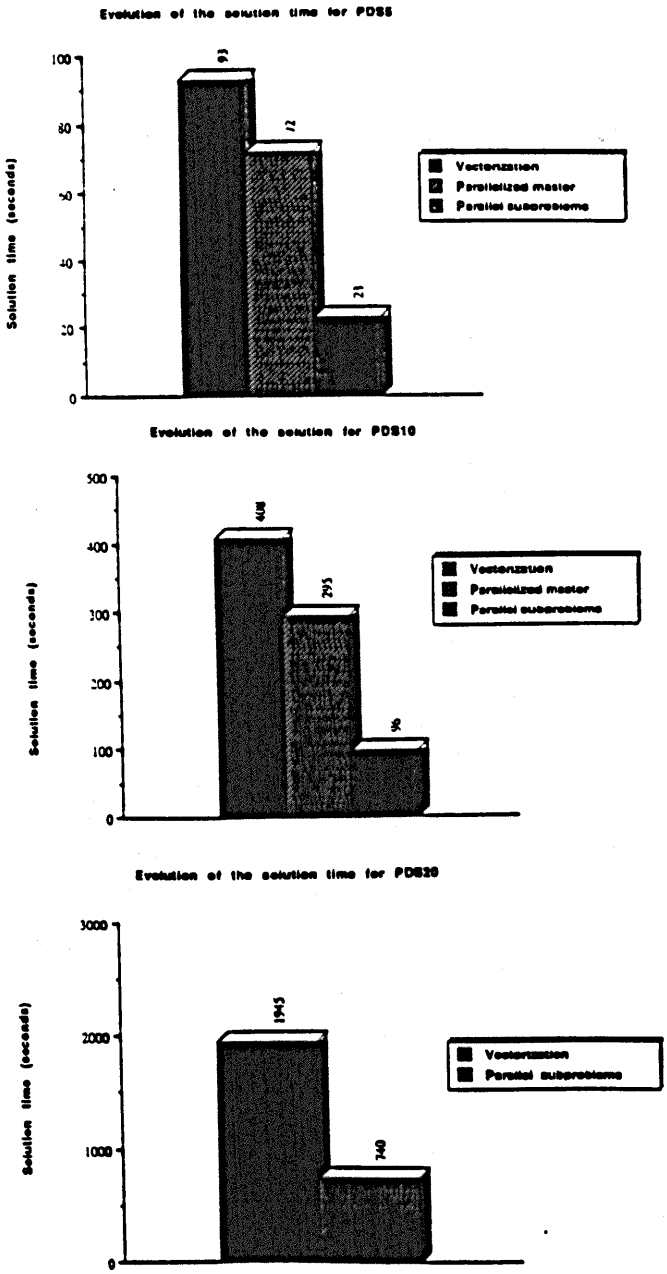


Figure 5.8: Evolution of solution times.

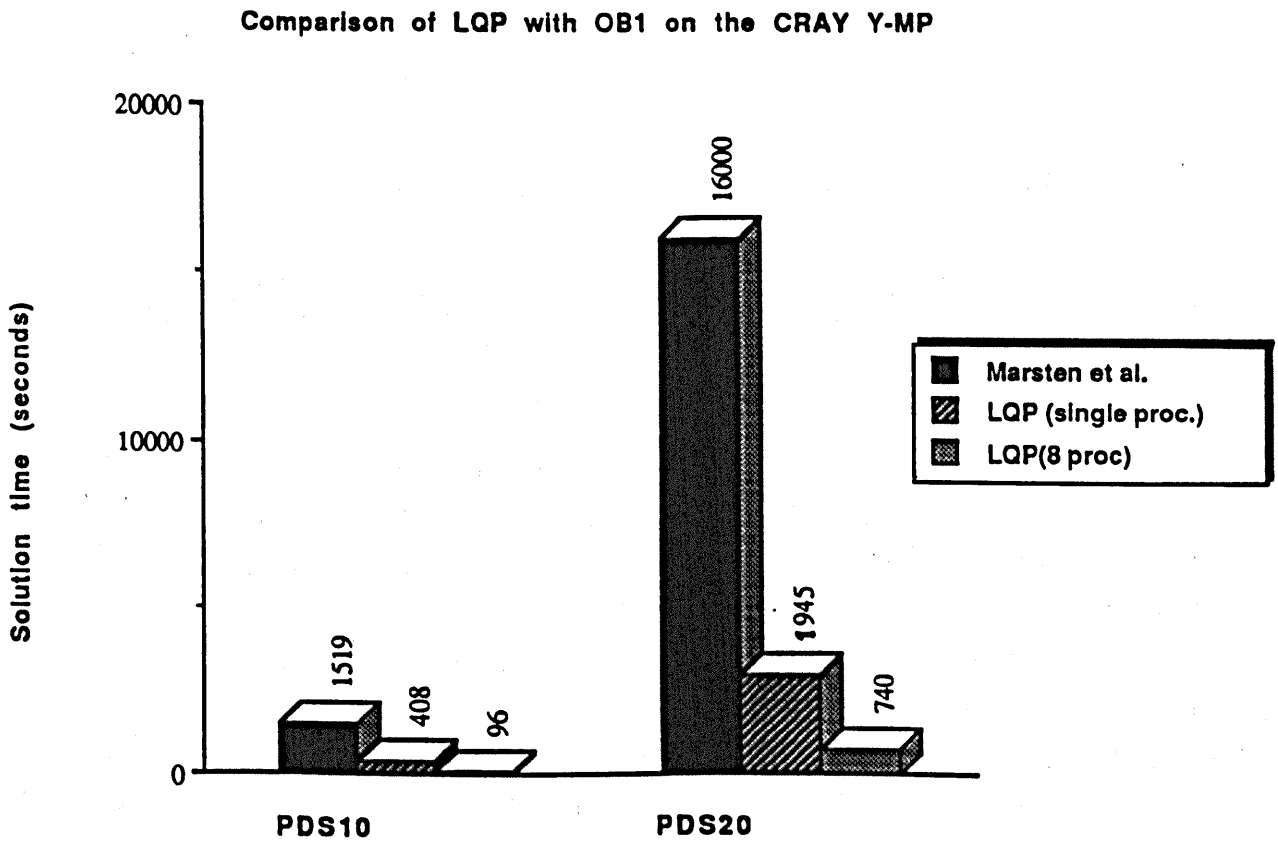


Figure 5.9: Comparison of the parallel decomposition with OB1

Chapter 6

Data-Level Parallelism for the Solution of Multicommodity Flows

We have developed in Chapter 5 a coarse-grain parallel implementation of the LQP algorithm on a CRAY Y-MP supercomputer. The parallel decomposition ideas were applied at both the linear algebra computations of the master problem and the parallel solution of linear network subproblems. The MIMD architecture of the CRAY Y-MP constitutes an ideal computing environment especially for the parallel solution of the linear network subproblems by multiple processors. In this scheme, each network subproblems were solved using the network simplex algorithm on multiple processors independently. This parallel computing scheme along with the vector and parallel computations at the master problem led to the coarse-grain parallel decomposition of the multicommodity network flow problems. In this chapter, we propose and develop a *massively parallel* or data-level parallel implementation of the LQP algorithm on a Connection Machine system. The following issues arise in this context:

1. The Connection Machine system is based on a SIMD (Single Instruction Multiple Data) parallel computing paradigm which is fundamentally different than the MIMD CRAY Y-MP system.

2. The Connection Machine offers up to 64K (K = 1024) processing elements and also the Virtual Processing (VP) scheme which allows the application programs to run on any number of processors without any changes, a property known as *scalability*.

The availability of a large number of tightly coupled processing elements makes the solution of the linear network subproblems a challenging task.

Our interest in a Connection Machine implementation stems not only from an effort to conduct a comparative study between the CRAY Y-MP and the Connection Machine but also from the belief that the Connection Machine is still a largely unexplored and new platform for scientific computing. Furthermore, the Connection Machine is built upon a parallel computing philosophy that is fundamentally different from the control-level parallelism of the CRAY Y-MP system. The Connection Machine is a *data-level* parallel system which associates one computing element with each data element. Thus, the same computation is executed at the same time on multiple data whereas on the CRAY system, multiple processors can execute different instructions at the same time. In this respect, the parallellism built into the Connection Machine offers a bigger challenge for the designer of algorithms. This statement is particularly true for optimization algorithms which are inherently sequential. We attempt in this section to build a software system on the Connection Machine to solve large scale multicommodity network flow problems.

We proceed with a brief exposition of data-level parallelism on the Connection Machine CM-2 system. We then try to identify the computational modules in the linear-quadratic penalty algorithm which can be mapped to the CM-2 system. We observed in Chapter 5 that the master problem is accountable for a large percentage of the computation time in the linear-quadratic penalty algorithm. The master problem computations are essentially dense linear algebra calculations as we have seen in Chapter 5 and can also be performed on the Connection Machine. Again, let us recall that the subproblem phase consists of solving linear network flow problems for each commodity. The massively parallel execution of the subproblem phase constitutes a more challenging issue. This is largely due to the limitations imposed by the lack of algorithms suitable for the Connection Machine architecture customized to solve linear network flow problems. Furthermore, no algorithm yet exists that can solve linear problems on the Connection Machine and compete with existing state-of-the art software such as OB1 of Marsten et al. [1990]. Network optimization has been one of the areas in optimization literature that benefited the most from the design of specialized algorithms for massively parallel architectures, see for instance Zenios and Lasken [1988], Nielsen and Zenios [1991b]. However, until recently these efforts were limited to the solution of problems with a strictly convex objective function. In Nielsen and

Zenios [1991b], this limitation is removed by adding a strictly convex nonlinear proximal term to the linear objective function and solving a sequence of such proximal subproblems to get a solution to the linear network flow problem. This is the approach we follow in this study to solve the linear network problem that arise in the subproblem phase. The strictly convex proximal subproblems are solved using a row-action type algorithm, Censor and Lent [1981], Nielsen and Zenios [1991b].

The rest of this chapter is organized as follows. We give a brief overview of data-level parallelism on the Connection Machine in section 6.1. In section 6.2 we discuss data-level parallelism for the master problem and present computational results with a set of Patient Distribution System Problems. Section 6.3 is devoted to a discussion of the subproblem phase on the Connection Machine and a summary of preliminary computational results.

6.1 An Overview of Data-level Parallelism on the Connection Machine System

Data-level parallel computing associates one processor with each data element of the problem at hand. This computing style exploits the natural computational parallelism inherent in many problems that deal with homogeneous operations on large data sets. Dense linear algebra computations are one of the areas in numerical computing where much research is devoted to the development of data-level parallel algorithms. The key to the implementation of an algorithm on a data-level parallel architecture is the layout of the problem data on some user specified configuration of the processing elements (PE)'s or processors. The algorithm is then executed by multiple PEs operating on local data. When there is interaction among the problem data such as an aggregation/coordination step, it becomes necessary to communicate among the processors through the prespecified configuration. The data-level parallel system used in this study is the Connection Machine CM-2 due to Hillis [1985].

The CM-2 is a fine-grain SIMD system. Its basic component is an integrated circuit with sixteen processing elements and a *router* that handles general communication. A fully configured CM-2 has 4,096 chips for a total of 65,536 PEs. The 4,096 chips are interconnected as a 12-dimensional hypercube. Each processor has local RAM of 32Kbytes, and for each cluster of 32 PEs a floating point accelerator handles floating point arithmetic. The Connection Machine provides the mechanism of *virtual processors* (VPs) that allows one

physical PE to simulate a number of virtual processors. The ratio of the number of virtual processors to the number of physical processors is referred to as the *VP ratio*. Operations by the PEs are under the control of a *microcontroller* that broadcasts instructions from a front-end computer simultaneously to all the elements for execution. A flag register at every PE allows for no operations.

Data parallel programming languages are available to allow the user to organize data so that program operations may be applied to many data elements at once. The programming languages for the Connection Machine system provide parallel processing without requiring the programmer to indicate synchronization explicitly in programs. The languages currently supported for the Connection Machine system are CM Fortran, C*, *Lisp. CM Fortran for the Connection Machine system is standard Fortran 77 supplemented with array-processing extensions. These extensions provide convenient syntax and numerous intrinsic functions to manipulate arrays. The array extensions are used to express efficient data parallel algorithms for the CM. CM Fortran also defines a set of intrinsic functions that take an argument and constructs a new array or a scalar. All these transformational functions take only array objects and all are therefore computed in parallel on the CM. We make extensive use of the array transformation functions in this study. A set of linear algebra subroutines is also available under the CMSSL (Connection Machine Scientific Subroutine Library) and some of these subroutines are essential to our implementation.

6.2 Master Problem Computations

Computation of the Descent Direction.

The most computing effort in the solution of the master problem is expended when solving the system:

$$(D^T H D)p = -D^T g \tag{6.1}$$

The symbols H and g are used respectively to denote the Hessian matrix and the gradient vector of the objective function $\Phi_{\mu,\epsilon}$ to simplify notation. The product $D^T H D$ can be computed in a straightforward by computing first the product $Y = HD$ and and the product $D^T Y$ using the CMSSL matrix multiplication routine. However, this would require knowledge of the entire matrix H whereas this is not necessary due to the following observation. The second derivative matrix H has a special structure when f is a linear function. Only

n entries of the $Kn \times Kn$ Hessian matrix need to be computed since all remaining entries are identical and equal to the first n. This is illustrated in Figure 6.1. Having made this observation, the computation can be performed as follows.

The product $C = D^T H D$ is computed on the Connection Machine in two phases. In the first phase the product $Y = HD$ is computed. In the second phase, the product $C = D^T Y$ is computed using the matrix Y from the first phase. As the computation of $D^T Y$ is rather straightforward to implement on the Connection Machine using the CMSSL routine GEN_MATRIX_MULT we will focus on the first phase computations. We think of each matrix as a two-dimensional object with individual elements laid out on a rectangular grid. The axis extents of the grid are given by the respective dimensions of the matrix. Each individual element of the matrix is associated with one node of the grid. Two matrices having the same shape (dimensions) are laid out so as to have their corresponding entries on each node (processor) of the rectangular grid. This scheme allows the same computation involving all elements of the two matrices to be performed in one step. In the light of these ideas, the first phase product Y is computed as follows. Let us recall that D is $Kn \times v-1$ matrix.

1. Spread the vector H "on" D along the x-axis $v-1$ times and along the y-axis K times, i.e., H is translated into conformable dimensions $(Kn \times v-1)$ as D

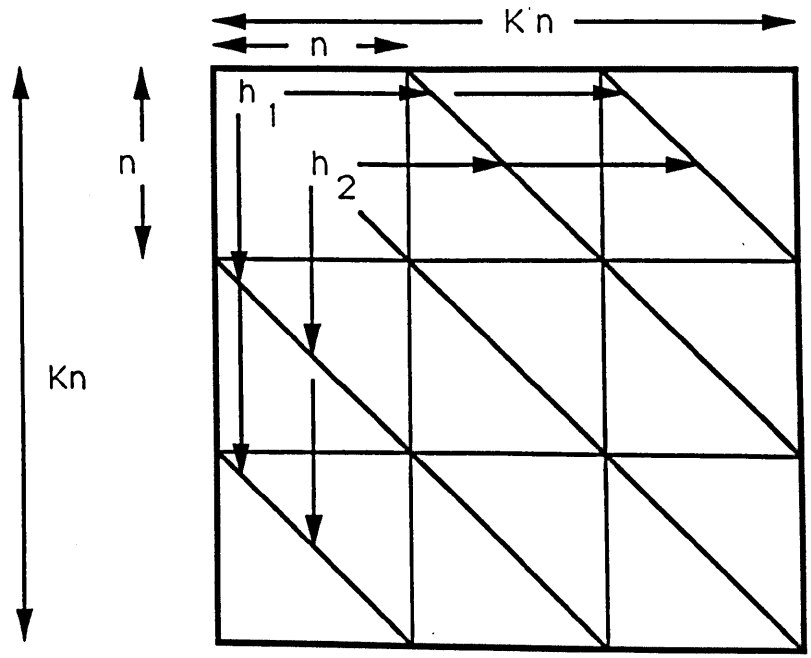
2. Perform elementwise multiplication, i.e., each processor (i,j) has access to elements H_{ij} and D_{ij} and performs the product $H_{ij}D_{ij}$ and stores the result. The result $\tilde{Y}_1$ is another $Kn \times v-1$ matrix with K slices along the y-axis.

3. Accumulate a sum by superposing each slice of the matrix $\tilde{Y}_1$ to obtain a matrix $\tilde{Y}_2$ of dimension $n \times v-1$.

4. Replicate the matrix $\tilde{Y}_2$ K times along the y-axis to get $Y = HD$.

The corresponding CM Fortran code is given below.

```
c
c -- ncom = number of commodities
c -- nvm1 = number of vertices - 1
c
```



The SPREAD and REPLICATE operation for the Hessian Matrix H. Only n entries are stored.

Figure 6.1: The Structure of the Matrix H.

```
      nvm1 = nv - 1
c
c -- spread H along D and perform multiplication
c
      d = d * spread(replicate(h_cm,dim=1,ncopies=ncom),
     $     dim=2,ncopies = nvm1)
c
c -- perform summation
c
      do 100 i=1,ncom
      dh2 = dh2 + d((i-1)*na+1:i*na,:)
 100  continue
c
c -- replicate to get Y
c
      d = replicate(dh2, dim=1, ncopies = ncom)
c
c -- get C
c
      call gen_matrix_mult(c,dt,d,1,2,ier)
```

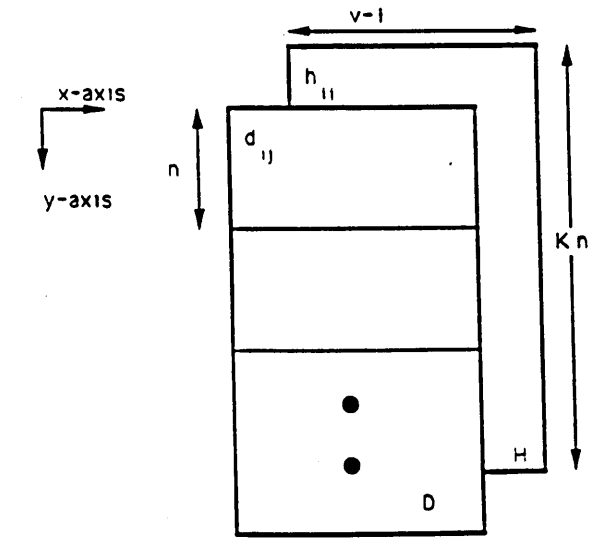
The matrix C is computed using the D^T and Y and the CMSSL routine GEN_MATRIX_MULT. We give a pictorial description in Figure 6.2.

The Computation of the Reduced Gradient.

The right-hand side of the linear system (6.1) can be computed on the Connection Machine. The matrix D and the vector g are both stored on the CM and the computation is performed using the CMSSL routine CM_MATRIX_VEC_MULT.

Function and Gradient Evaluations.

The function and gradient evaluations can be performed on the Connection Machine in a straightforward manner. First, we compute the linear objective function as a dot product of the cost vector and the current iterate. Next, we compute the function value contributed by



Add all K slices of the above elementwise product to get Y

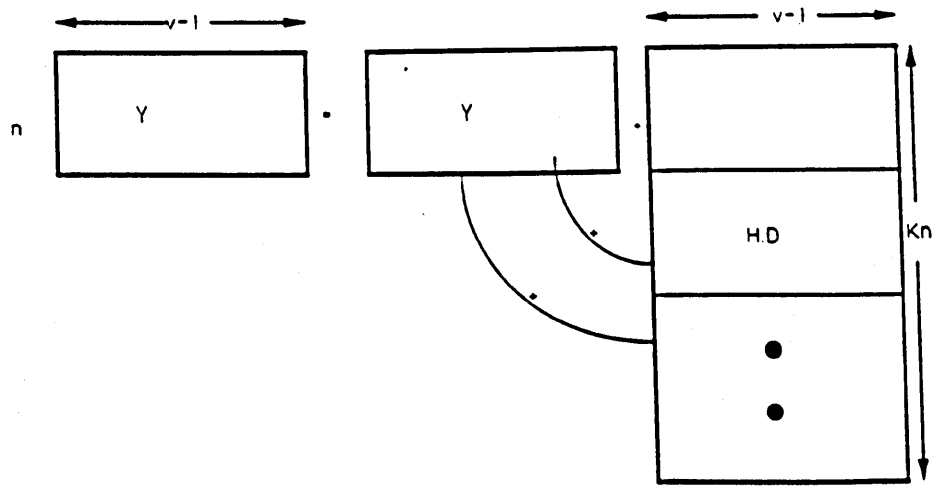


Figure 6.2: Computing the product HD on the CM

the penalty function. A conditional **WHERE** construct computes the quantity contributed by each side constraint (joint capacity constraint) evaluated at the current iterate. The violation in the side constraint inequalities are computed prior to the call to the function and gradient evaluation routine.

The CM Fortran code is given below.

```
c
c -- nlen = number of arcs * number of commodities
c
c -- above.. array where side constraint values are stored
c            0 if constraint is satisfied.
c
      nlen = na*ncom
c
c --- linear objective function computed as a dot product
c
      obj_cm = dotproduct(x1_cm,cost_cm)
c
c --- the nonseparable penalty function terms
c
c
      gradient = 0.0
c
      where (above .gt. epsi)
c
c -- constraint in the linear region
c
      above = pmu*(above - epsi/2)
      gradient =  pmu
      elsewhere
c
c -- constraint in the quadratic region or satisfied
c
```

```
      gradient =  pmu*(above/epsi)
      above = pmu*(above**2)/(2*epsi)
      end where
c
c -- the value of the objective function =
c
c     linear part + penalty terms
c
      obj_cm = obj_cm + sum(above)
c
      g_cm = cost_cm + replicate(gradient, dim=1,ncopies=ncom)
```

The Computation of the New Iterate.

We compute the new iterate x based on the simplicial weights as

$$x = Bw. \tag{6.2}$$

Again, this is a fully dense matrix-vector product that is performed on the CM using the CMSSL routine CM_MATRIX_VEC_MULT.

Numerical Results and Discussion.

A set of numerical results using the Patient Distribution System (PDS) problems are presented in this section. We give in Table 6.1 solution statistics on a 32K CM-2. The components of master problem solution time on the SUN Workstation Front End and the CM-2 are plotted in Figure 6.3 for PDS5. The notation Bw stands for the computation of the new iterate x based on the updated weights w, Dg stands for the computation of the right=hand side of the system (6.1), LS stands for the linesearch function and gradient evaluations. The distribution of the total solution time between the subproblem and the master problem are on the CM and on the front end Sun Workstation is illustrated in Figure 6.4. We also plot the evolution of solution times on the SUN Front End, on a CM-2 with 8K processing elements and on a CM-2 with 32K processing elements as a function of problem size in Figure 6.5. The rate of growth of the solution on the 32K CM-2 seems to be much lower than the rate of growth on the Front End, a positive impact of parallelism. The CM Fortran code was compiled with `-slicewise` option. The computation was stopped

when the degree of infeasibility in the joint capacity constraints was less than 10^{-1} to economize on Connection Machine resources. However, the objective function value reported on termination has four to five digits of accuracy compared to the optimal values reported by MINOS. The times reported are estimates of wall clock time in seconds.

Test Problem	Subproblem time	Master problem time	Total time
PDS1	4.8	21.0	25.8
PDS3	52.33	127.77	179.1
PDS5	252.0	426.0	678.0
PDS10	1372.0	2224.0	3596.0

Table 6.1: Performance of the linear-quadratic penalty algorithm on a 32K CM-2 system with master problem computations on the CM.

We observe that with respect to solution time these results do not compare favourably with the computational results obtained on the CRAY Y-MP. The parallel execution of the master problem phase had resulted in a reversal of the percentage share of the subproblem time and the master problem time. We noted that the master problem was accountable for a much larger percentage of the solution time in a serial computing environment. However, the subproblem time was the dominating factor after the parallelization of the master problem phase. This situation is not observed in our Connection Machine implementation and the master problem time still accounts for a larger percentage of the total runtime. The main reasons for this behaviour are:

- The "thin" shape of the matrix D which affects the performance of matrix-matrix and matrix-vector operations on the CM. Best performance results are achieved with square matrices on the Connection Machine.

- Communication costs are incurred during the **SPREAD** and **REPLICATE** operations. The SPREAD operation takes as argument an array and creates another array with an extra dimension. The REPLICATE operation takes an array and replicates the entries along a given dimension. Both these operations are used extensively in the descent direction computation module and result into interprocessor communication.

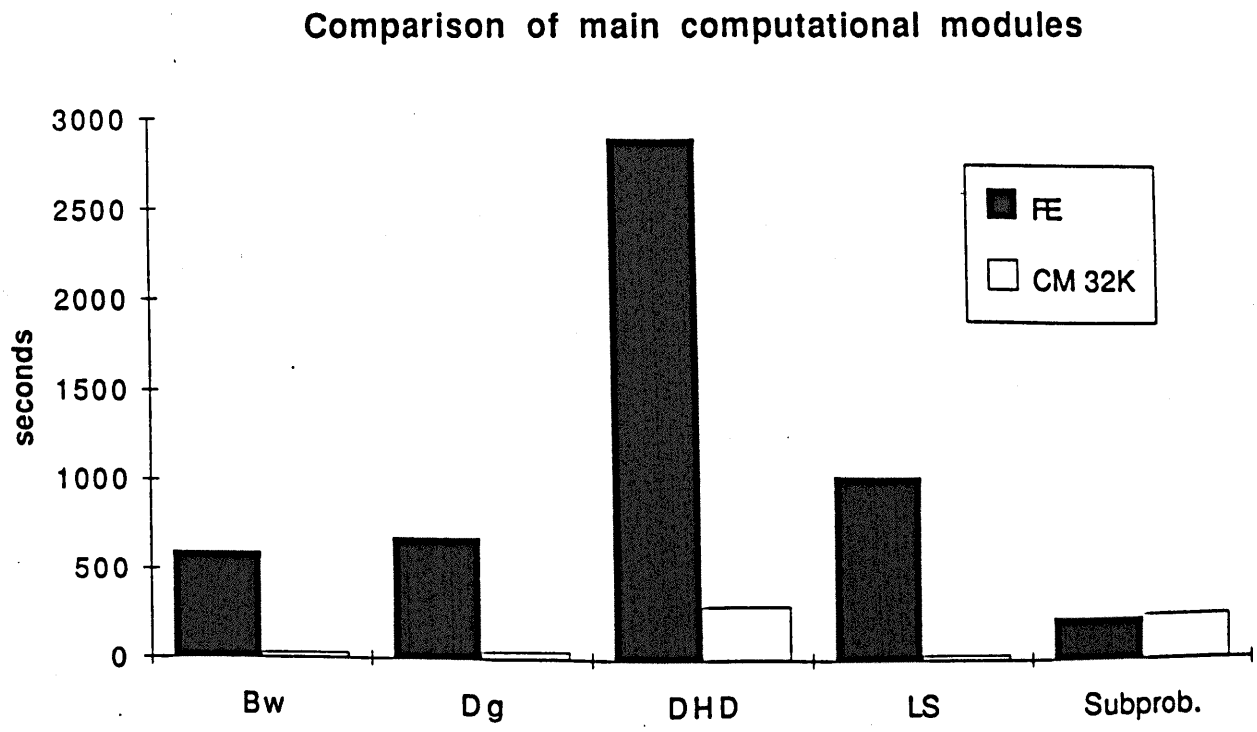


Figure 6.3: The Components of the Master Problem on the Front End and on the CM

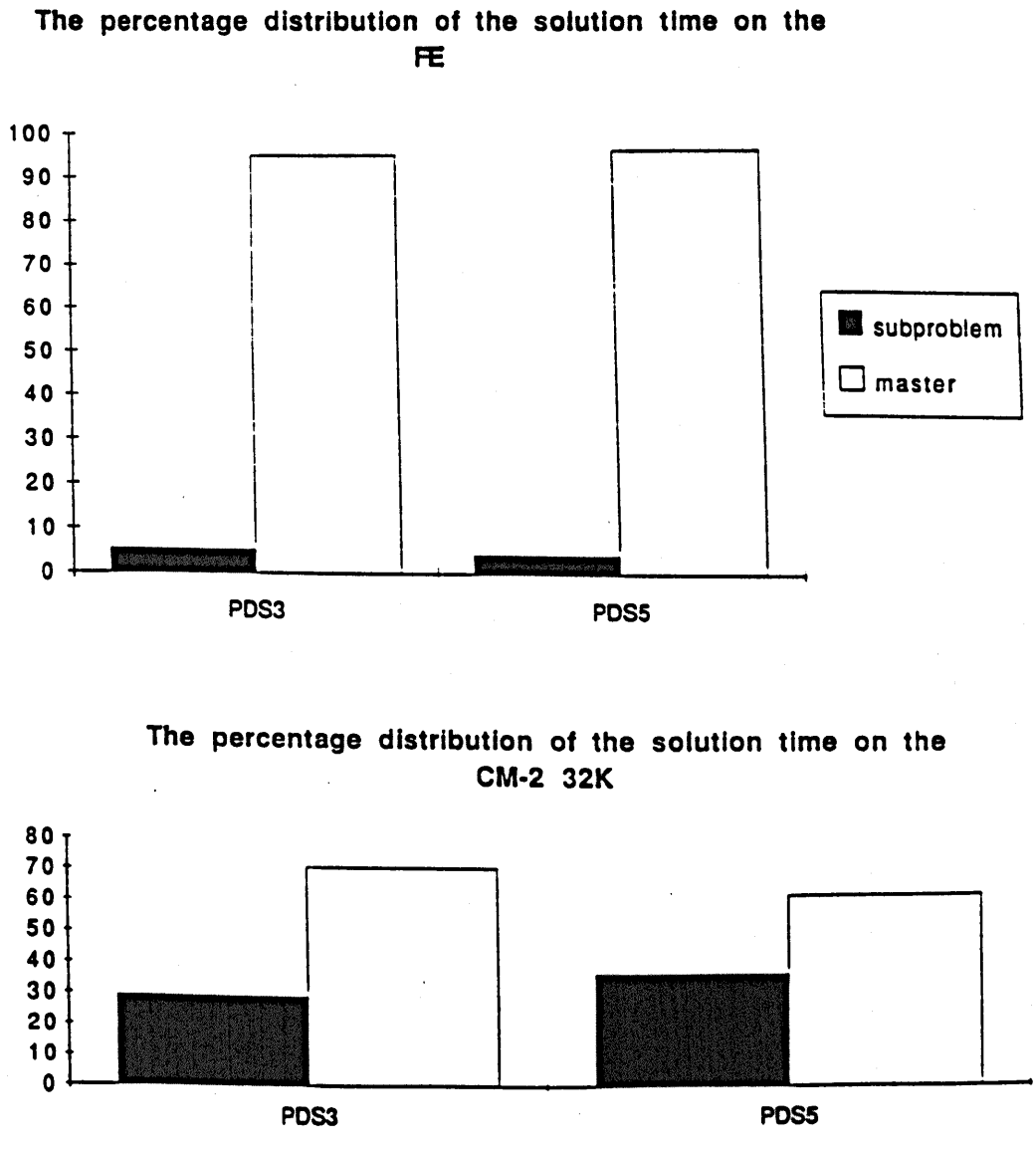


Figure 6.4: The Distribution of Solution time between the Subproblem and the master problem on the Front End and on the CM

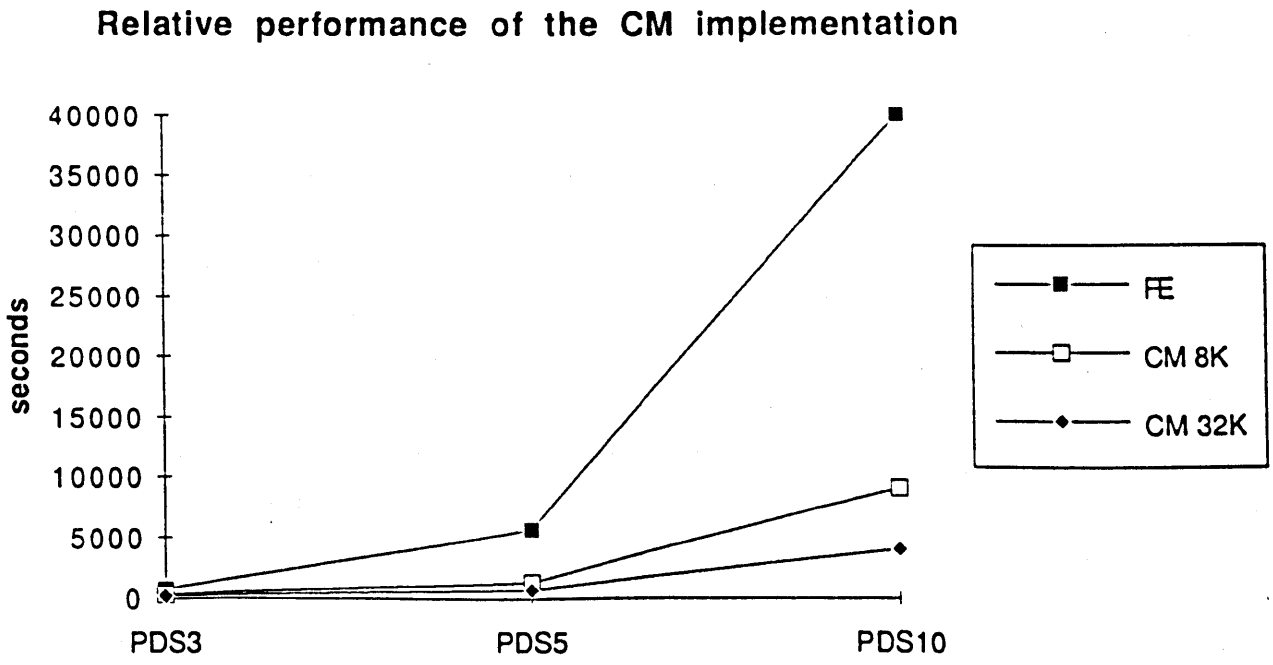


Figure 6.5: The Evolution of Solution Times

- Moving data between CM arrays of different sizes also results in interprocessor communication which affects performance negatively.

Nevertheless, as a first attempt to build a solver for multicommodity flows on a data-level parallel computer system, the results are still encouraging. We also remark that Zenios [1991] developed a specialization of the row-action algorithms of Censor and Lent [1981] to solve quadratic multicommodity transportation problems on the Connection Machine. Although our algorithm is not a "truly" massively parallel algorithm as the row-action type algorithms, it is one of the first general purpose decomposition methodology implemented on the Connection Machine and, as such, removes the strict convexity requirements stated in Zenios [1991] by enabling the solution of linear programs.

6.3 Subproblem Computations

As we remarked earlier, the subproblem phase consists of solving linear network flow problems. This phase is performed on the Connection Machine. A nonlinear proximal term is appended to the linear objective function resulting into a strictly convex program. This nonlinear strictly convex network program is solved using the row-action algorithm of Censor and Lent [1981] specialized for network flow problems in Zenios and Censor [1991].

We proceed with a brief overview of the proximal minimization algorithm. The main reference for the material summarized in this section can be found in Nielsen and Zenios [1991b].

6.3.1 The Proximal Minimization Algorithm with D-functions (PMD)

The PMD algorithm was proposed in Censor and Zenios [1991], where its convergence was established. Let $S \neq \emptyset$ be an open convex set. Let $f : \Lambda \subseteq \Re^n \mapsto \Re$ be an auxiliary function. We assume that $\bar{S} \subseteq \Lambda$, where $\bar{S}$ is the closure of S, and that f is strictly convex and continuous on $\bar{S}$ and continuously differentiable on S. The set S is called the *zone* of f. We define the *D-function* corresponding to f as

$$D_f(x, y) = f(x) - f(y) - \nabla f(y)^T(x - y). \tag{6.3}$$

Consider now the linear network flow problem

$$\min_{x \in X} c^T x, \tag{6.4}$$

where $X \subset \Re^n$ is the polyhedral feasible region, $X = \{x \in \Re^n \mid Ax = b, 0 \leq x \leq u\}$, assumed to be nonempty. A is the usual node-arc incidence matrix, see, e.g., Kennington and Helgason [1980]. For some suitable choice of the auxiliary function f and a positive sequence $\{\gamma^k\}_{k=0}^{\infty}$ with $\liminf_{k\to\infty} \gamma^k = \gamma < \infty$, the **proximal minimization algorithm with D-functions** proceeds from an arbitrary starting point, $x^0 \in S$, with the following iteration.

The PMD Algorithm.

$$x^{k+1} \leftarrow \arg\min_{x \in X \cap S} c^T x + \frac{1}{\gamma^k} D_f(x, x^k). \tag{6.5}$$

The convergence of the PMD algorithm to a solution of the minimization problem (6.4) under some appropriate choice of the auxiliary function f was proved in Censor and Zenios [1991]. In particular, f should be a Bregman's function, as defined by Censor and Lent [1981], and its zone S should satisfy $X^* \cap \bar{S} \neq \emptyset$, where X^* is the optimal set, $X^* = \{x^* \in X \mid c^T x^* \leq c^T x, \forall x \in X\}$.

Quadratic Proximal Terms

Consider now the PMD algorithm with the auxiliary function:

$$f(x) = \frac{1}{2} \| x \|^2, \tag{6.6}$$

where $\| \cdot \|$ denotes the Euclidean norm. The D-function induced by (6.6) is then $D_f(x, y) = \frac{1}{2} \| x - y \|^2$. We have in this case $\Lambda = S = \bar{S} = \Re^n$, and we obtain as a special case of (6.5) the **quadratic proximal point algorithm (QPP)** of Rockafellar [1976a, 1976b]:

The QPP Algorithm.

$$x^{k+1} \leftarrow \arg\min_{x \in X} c^T x + \frac{1}{2\gamma^k} \| x - x^k \|^2 . \tag{6.7}$$

We mention that, for linear programs, QPP is finitely convergent. Also, there exists an $\bar{\gamma} < \infty$ such that for any $\gamma^k \geq \bar{\gamma}$ and any fixed x^k, the solution to the minimization in (6.7) is also a solution to (6.4) (see, e.g., Bertsekas and Tsitsiklis [1989, Section 3.4.3]).

6.3.2 Solving the Strictly Convex Network Programs

We now turn to the solution of the inner minimization problem at each step of the quadratic proximal minimization algorithm.

The inner minimization algorithm is a primal-dual, row-action algorithm. Dual feasibility and complementary slackness are maintained throughout, but primal feasibility is obtained only in the limit. The driving force of the algorithm is the dual variables, one for each network node. Due to the strict convexity of the objective function, given a set of dual variables, the primal (flow) variables satisfying complementary slackness are uniquely given. The algorithm operates at one constraint (corresponding to a network node) of the constraint set at a time. It computes a primal-dual pair so to maintain complementarity to satisfy the chosen constraint exactly. It can be described as follows:

Step 0: (Initialization) Given is an initial set of dual prices, π_i^0, for each node i. Let $k \leftarrow 0$.

Step 1: (Update flows) Compute for each arc (i,j) the unique flows x_{ij}^k which, together with the dual prices π_i^k, satisfy complementary slackness.

Step 2: (Compute surplus) For each node of the network, compute the flow surplus or deficit based on the flows x_{ij}^k of the incident arcs, and the node supply or demand (i.e., compute $(Ax^k - b)_i$ for each node i). If the resulting surplus or deficit for each node is within a tolerance of zero, terminate with an approximately feasible solution satisfying complementary slackness and dual feasibility.

Step 3: (Update prices) Update for each node the dual price in a direction which will decrease the subsequent surplus or deficit. Set $k \leftarrow k + 1$ and proceed from Step 1.

For a complete description of the algorithm, see Bertsekas and Tsitsiklis [1989, Section 5.5], or Zenios and Lasken [1988]. The crucial step of the algorithm is the price update in Step 3, and there are different ways to do this, as discussed in Nielsen and Zenios [1991a].

The row-action algorithm considers one node at a time. The primal variables representing flows on arcs incident to the node are changed to the "closest" values which also satisfy the flow conservation constraint for the node. The dual price update is such as to maintain complementary slackness. The measure of "close" used in the row-action algorithm is given by the D-function induced by the network problem objective function. This update is called a *Bregman projection.*

6.3.3 Numerical Results and Discussion

The quadratic proximal point/row-action (QPPRA) algorithm of Nielsen and Zenios [1991b] is used to solve the linear network subproblems on the Connection Machine. To solve the linear network problems, the QPPRA code is called as a subroutine. Since QPPRA was written in C/PARIS (Parallel Instruction Set), a CM Fortran C/PARIS interface was build to link the LQP code with the QPPRA solver. The computational results are summarized in Table 6.2. We report the total time to solve each test problem.

Test Problem	Total time
PDS1	8 mins.
PDS3	43 mins.

Table 6.2: Performance of the linear-quadratic penalty algorithm on a 16K CM-2 system with both master problem and subproblem computations on the CM.

Due to the anticipated CPU usage, the remaining PDS problems were not tested. It is clear that it is more advantageous to solve the linear network subproblems using the network simplex method on the Front End. Whether the subproblems can be solved in parallel using multiple processors while the master problem computations are performed on the CM remains a research question. One encouraging point is that the using the QPPRA solver does not impair the progress the LQP algorithm in terms of reducing the degree of infeasibility in the joint capacity constraints and the number of iterations to termination. In Figure 6.6, we plot the infeasibility error versus the number of major iterations (1) when the network simplex was used on the Front End (2) when the QPPRA replaces the network simplex solver.

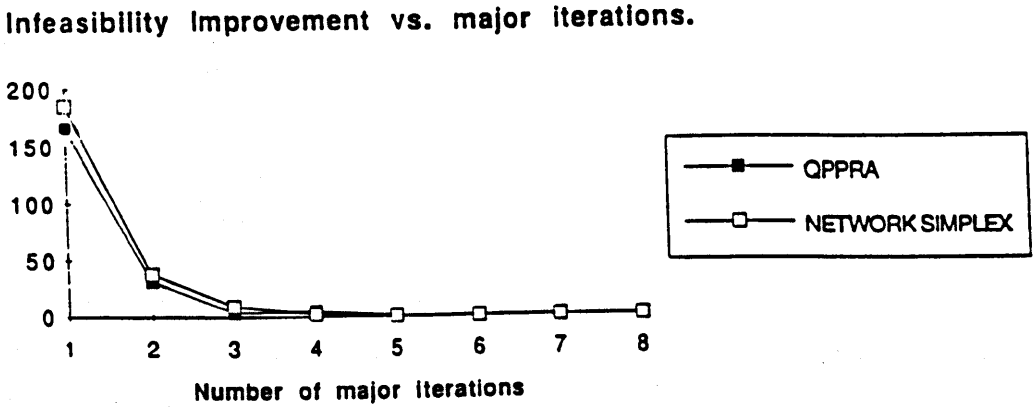


Figure 6.6: Infeasibility Improvement with 1. QPPRA 2. Network Simplex for PDS1

Chapter 7

Naval Personnel Assignment: An Application of Linear-Quadratic Penalty Methods

7.1 Introduction

The use of specialized algorithms can have a significant impact on the solution process of large scale optimization models. This is particularly true for models which are used for operational planning purposes where solutions need to be produced in a timely and cost-effective way. The use of special purpose algorithms could make the difference between a conceptual model and one that is routinely used to aid the decision making process. Even for strategic planning applications, however, mathematical programming has evolved into a powerful modeling tool in various application areas such as transportation, logistics and military planning among many others. These models can be solved using general purpose mathematical programming packages. However, these models tend to be very large in general and require excessive amount of time and storage during the solution process. In such cases, it may be beneficial to exploit any structure present in the model to produce a solution within reasonable time and memory usage. This strategy has been followed in the use of network structures for planning purposes since the early days of linear programming. See for instance the survey papers by Glover, Klingman and Phillips [1990] and by Dembo,

Mulvey and Zenios [1989].

In this chapter we describe a military planning model which was formulated as a very large linear program with an embedded network structure. The problem is solved by a specialization of a smooth penalty algorithm.

This chapter is organized as follows. In section 7.2 we describe the Naval personnel assignment problem and the associated network formulation. A more detailed account can be found in Krass and Thompson [1990]. In section 7.3 we specialize the (LQP) algorithm to the Naval personnel assignment model. Section 7.4 contains an account of solution strategies and numerical results. We also provide comparisons with the general purpose linear program solver MINOS of Murtagh and Saunders [1987]. We conclude the chapter in section 7.5 with general comments and discussion.

7.2 The Naval Personnel Assignment Problem

Each year thousands of decisions are made to (re)allocate the Navy Enlisted Personnel to a fleet of combat units and to mission areas within these units. Allocations are made in such a manner as to provide the best defense at the lowest cost. A combat unit is characterized by a number of mission areas according to the function fulfilled by each mission area such as mobility, anti-air warfare, submarine warfare etc. A mission area within a unit requires personnel with different skills to support operational capabilities. The personnel skills can be determined by ranks, pay grades and the Navy Enlisted Classifications. A unit's capability to perform its functions in all its mission areas is referred to as "readiness". Manning is defined as the percentage of personnel to manpower requirements of a mission area. Readiness is measured based on the manning level of a mission area. A shortage of skilled personnel would decrease the level of readiness of a mission area and thus degrade the capabilities of the unit. Clearly maximizing the level of readiness is a complex decision making problem given the large number of mission areas and personnel to be matched. Several researchers have studied this problem and suggested models and solution techniques. Network models for general military personnel planning have been proposed by Klingman, Mead and Phillips [1984], and for the Navy problem by Ali, Kennington, Liang [1988].

In this chapter we are particularly interested in the Navy personnel planning problem, under considerations of readiness. Quantifying the level of readiness of a unit as a function

of the personnel assigned to the unit becomes a crucial first task in the decision making process. A continuous measure of readiness was recently developed in Krass, Liang and Thompson [1988]. They also present a heuristic that computes the number of personnel moves required to achieve a given level of readiness for each unit. Classifying the personnel in two categories they measure the readiness level R for a mission area as:

$$R = 10 - \frac{\min[(x+5), y]}{10} \tag{7.1}$$

where x is the percentage of personnel from category A and y is the percentage of personnel from both categories A and B. Smaller values of R indicate a high level of readiness for a mission area. The readiness level for a unit is defined as the minimum of the readiness levels of all mission areas contained in that unit. Therefore given the number of Navy personnel from each category assigned to a unit u and to each mission area m within unit u, the readiness level of the unit can be computed as

$$R_u = \max_m R_{um} \tag{7.2}$$

where R_{um} is obtained from (7.1). The problem of assigning personnel to units and to mission areas within units ignoring readiness considerations can naturally be modeled as a network optimization problem, see Krass and Thompson [1990]. An example is depicted in Figure 7.1. We sketch here the network underlying the optimization model and the additional requirements that form the non-network constraints. The network consists of three layers:

Layer 1: Navy personnel divided into categories according to their skills, rank and pay grades. Each such category constitutes a supply node, with supply equal to the number of personnel available for assignment in each category. These nodes are connected to second layer nodes.

Layer 2: The Combat units. There is a node for each unit to which personnel are to be assigned. These are transshipment nodes, i.e., they have no exogenous supply or demand. These nodes are connected to third layer nodes.

Layer 3: The Mission areas. There is a node for every mission area of each unit. These nodes have supply equal to the number of personnel already on board a given unit mission area.

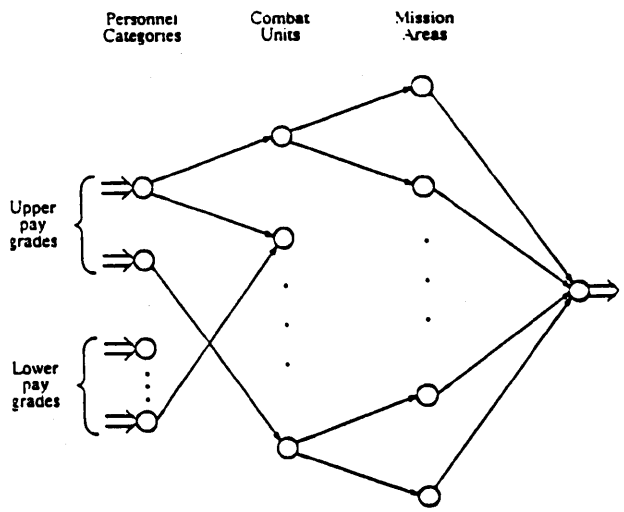


Figure 7.1: The structure of the network for Naval Personnel Assignment

Finally these nodes are connected to a sink node with demand equal to total personnel supply. The network can be seen as a tripartite graph. The model takes into account the following additional considerations:

- A person assigned to a unit may or may not be counted toward the readiness level calculation of some mission areas within the unit. Clearly if a person cannot perform some function critical to the mission area then his or her contribution to the readiness level for that area is nil. This requirement depends on the problem data and can be modeled using side constraints.
- The readiness level computation is non-linear in the flow variables since it involves the "max" operator, see equation (7.2). This computation can be reformulated using linear inequalities and posed as non-network constraints. This reformulation is given below.

The above requirements are cast into side constraints as follows. Let

x_{rpu} be the number of personnel from pay grade p with skill r assigned to unit u,

Y_{pum} the number of personnel assigned to unit u that can be employed in mission area m.

s_{urm} a parameter which is equal to 1 if a person with skill r assigned to unit u can be employed in mission area m and zero otherwise. This parameter is specified as part of the problem data.

PB^p_{um} the number of personnel from pay grade p already on board unit u and that can be employed in mission area m.

B^p_{um} the total number of personnel from pay grade p that can be accomodated on board unit u and mission area m.

The parameters PB^p_{um} and B^p_{um} are also specified as problem data. The decision variables x_{rpu} and the variables Y_{pum} are related by the following relation:

$$Y_{pum} = \sum_r s_{urm} x_{rpu} \tag{7.3}$$

Assuming there are only two pay grades, i.e., the value of p is either 1 or 2, the readiness level R_u for unit u is computed as:

$$R_u = 10 \cdot (1 - \min_m(\min((\frac{Y_{1um} + PB^1_{um}}{B^1_{um}} + .05), \frac{Y_{2um} + PB^2_{um}}{B^2_{um}}))) \tag{7.4}$$

Then the problem of maximizing fleet readiness is to find the fleet readiness level R such that R is a solution to

Minimize $\quad R = \max_u R_u$

Subject to: Flow conservation constraints on x_{rpu}

This problem can be reformulated by observing that if L is a feasible readiness level then

$$L \geq R_u \quad \forall \ u$$

This condition is equivalent to

$$L \geq 10 \cdot (1 - \min_m(\min((\frac{Y_{1um} + PB^1_{um}}{B^1_{um}} + .05), \frac{Y_{2um} + PB^2_{um}}{B^2_{um}}))) \tag{7.5}$$

Defining $z = 1 - \frac{L}{10}$, then (7.5) can be rewritten as:

$$z \leq \min_m(\min((\frac{Y_{1um} + PB^1_{um}}{B^1_{um}} + .05), \frac{Y_{2um} + PB^2_{um}}{B^2_{um}}))) \tag{7.6}$$

or, equivalently, the readiness level L is attainable if

$$Y_{1um} \geq (z - .05) \cdot B^1_{um} - PB^1_{um} \tag{7.7}$$

and

$$Y_{2um} \geq z \cdot B^2_{um} - PB^2_{um} \tag{7.8}$$

Using (7.3) the conditions for measuring readiness level are equivalent to

$$\sum_r s_{urm} x_{r1u} \geq (z - .05) \cdot B^1_{um} - PB^1_{um} \tag{7.9}$$

$$\sum_r s_{urm} x_{r2u} \geq z \cdot B^2_{um} - PB^2_{um} \tag{7.10}$$

Rearranging terms, the side constraints have the following form:

$$\sum_r s_{urm} x_{rpu} - \alpha z \geq r \quad \forall \quad u, p, m \tag{7.11}$$

where the scalars α and r are computed from (7.9) and (7.10). Therefore, the Naval personnel assignment model can be formulated as:

Maximize z

Subject to: $\sum_r s_{urm} x_{rpu} - \alpha z \geq r \quad \forall \quad u, p, m$

$+$

Flow conservation constraints on x_{rpu}

Expressed in matrix form:

[NETSIDE]

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & -z \\ \text{subject to} \quad & Ax = b \\ & Sx + Pz \geq r \\ & 0 \leq x \leq u \\ & 0 \leq z \leq v \end{aligned}$$

where A is a node-incidence matrix for the network and represents the flow conservation conditions and S and P are matrices used to capture the non-network requirements. The

variables x denote the flow variables and z is the side variable which represents the level of readiness to be maximized. The parameters u and v are upper bounds on the flow variables and the side variable(s). The objective is to maximize the level of readiness for all units considered.

7.3 Application of the Linear-Quadratic Penalty (LQP) Method to Naval Personnel Assignment

In this section we describe the Linear-Quadratic Penalty (LQP) algorithm of Zenios, Pınar and Dembo [1990] as applied to the Naval personnel assignment problem. The development in this section is essentially identical to the material given in section 3.7 and is repeated here for the sake of completeness.

The linear-quadratic penalty function used throughout this thesis is used to eliminate the side constraints by placing those in the objective function. The nonlinear network problem obtained by penalizing the side constraints $Sx + Pz \geq r$ is formulated as:

[NLP]

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & \Phi(x,z) = -z + \mu \sum_j \tilde{p}(y_j) \\ \text{subject to} \quad & Ax = b \\ & 0 \leq x \leq u \\ & 0 \leq z \leq v \end{aligned}$$

where $y = r - Sx - Pz$, the linear-quadratic penalty function is given by (2.4) and μ is a positive scalar which determines the severity of the penalty. The resulting nonlinear network problem is solved repeatedly with adaptively changing parameters μ and ϵ until suitable stopping criteria are satisfied. The algorithm can be concisely stated as follows:

The Linear–Quadratic Penalty Algorithm

Step 0 (Initialization.) Find an initial feasible solution for the network component of Naval

Personnel Assignment problem ignoring the side constraints, i.e., solve the problem

$$\begin{aligned} \underset{(x,z)}{\text{minimize}} \quad & -z \\ \text{subject to} \quad & Ax = b \\ & 0 \le x \le u \\ & 0 \le z \le v \end{aligned}$$

If the solution to this problem satisfies all side constraints, stop. Otherwise choose initial values for penalty parameters μ and ϵ and go to Step 1.

Step 1 Solve the nonlinear network problem **NLP**. Go to Step 2.

Step 2 If the solution satisfies optimality criteria, stop. Otherwise, adjust the penalty parameters μ and ϵ and go to Step 1.

The solution of the nonlinear network problem in Step 1 demands the most computational effort. This problem is solved using the network specialized version of simplicial decomposition algorithm, see Mulvey et al [1990].

Simplicial decomposition iterates by solving a sequence of linearized subproblems to generate extreme points of the feasible region of the network component and master problems which minimize the nonlinear objective function over the simplex spanned by the extreme points. A complete description is given in the Appendix. Here we discuss the specializaton of the algorithm for the Naval personnel assignment model.

The Subproblem. A new vertex (x, z) is generated as the solution to the following subproblem:

$$\begin{aligned} \underset{x,z}{\text{Minimize}} \quad & x^T \nabla_x \Phi(x^\nu, z^\nu) + z^T \nabla_z \Phi(x^\nu, z^\nu) \\ \text{subject to} \quad & \\ & Ax = b \\ & 0 \le x \le u \\ & 0 \le z \le v \end{aligned}$$

where (x^ν, z^ν) is the iterate at the ν-th iteration of simplicial decomposition and $\nabla_x \Phi$ and $\nabla_z \Phi$ denote the gradient with respect to x and z respectively. This problem naturally

decomposes into two independent linear programs as follows:

$$\begin{aligned} \underset{x}{\text{Minimize}} \quad & x^T \nabla_x \Phi(x^\nu, z^\nu) \\ \text{subject to} \quad & \\ & Ax = b \\ & 0 \leq x \leq u \end{aligned}$$

and

$$\begin{aligned} \underset{z}{\text{Minimize}} \quad & z^T \nabla_z \Phi(x^\nu, z^\nu) \\ \text{subject to} \quad & \\ & 0 \leq z \leq v \end{aligned}$$

The first problem is a linear network problem and is solved using the network simplex method. The second problem is solved trivially by assigning z to its lower or upper bound depending on the sign of the gradient $\nabla_z \Phi(x^\nu, z^\nu)$.

7.4 Numerical Results

The LQP algorithm was implemented for the case of the problem NETSIDE. The code was written in Fortran. We refer to the code specialized for the Navy assignment problems as the GENOS/LP system. In this section we report numerical results obtained using GENOS/LP on two Naval Personnel Assignment problems. The first model — NAVY — is a simplified version of the complete model which we we call HUGENAVY. The size and characteristics of both problems are given in Table 7.1. The objective in both problems is to minimize the readiness measure as introduced in section 7.2 or equivalently to maximize the readiness of the fleet. Both problems have one side (non-network) variable which represents the readiness measure.

Problem	Linear Programming		Network		Number of
	Rows	Columns	Nodes	Arcs	side constraints
NAVY	4144	6842	3457	6841	687
HUGENAVY	36013	64542	30639	64541	5374

Table 7.1: Test Problem Characteristics.

7.4.1 Solution Strategies

Before we give the solution statistics for the HUGENAVY problem we mention two particularly important components of the GENOS/LP system.

For the Naval assignment problems used in this study, an initial feasible solution was readily available since the solution to the network relaxation satisfied the side constraints when the side variable was ignored, i.e., let x^0 be a solution of the network relaxation

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & 0x \\ \text{subject to} \quad & Ax = b \\ & 0 \leq x \leq u \end{aligned}$$

If x^0 is such that

$$Sx^0 \geq r$$

then z^0 is computed as

$$z^0 = \min_i \frac{(\sum_{ij} s_{ij}x_{ij} - r_i)}{p_{ij}} \tag{7.12}$$

where s_{ij} and p_{ij} denote the entries at the i-th row and j-th column of the matrices S and P respectively.

The computation is terminated when both of the following error measures are within acceptable tolerance.

1. Absolute error in side constraint feasibility

$$\|r - Sx - Pz\|_\infty \leq \epsilon_{min}$$

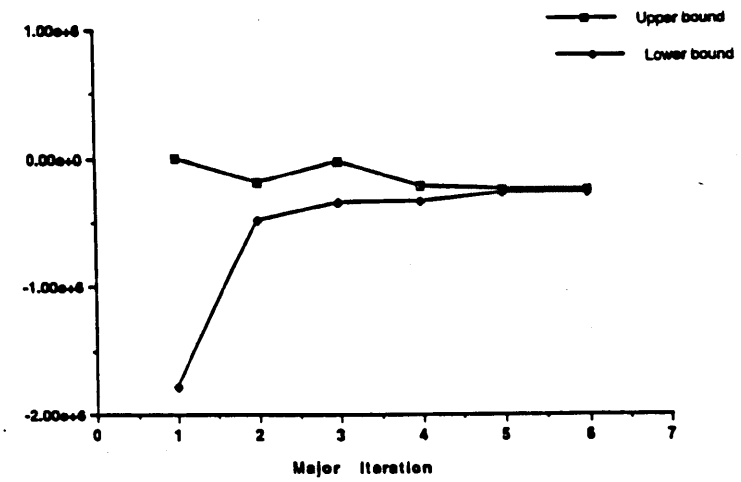


Figure 7.2: Convergence of lower and upper bounds for NAVY

2. Bound gap

$$\frac{-\underline{z} - h(\mathbf{x})}{h(\mathbf{x})} \leq \epsilon_{gap}$$

where $\mathbf{x} = (x, z)$ is the current iterate and $(\underline{x}, \underline{z})$ is obtained from (3.42) and h is the lower bound function discussed in Chapter 3. The values of ϵ_{min} and ϵ_{gap} used in this study are 10^{-5} and 3×10^{-2} respectively. The lower bound generated as a result of the procedure described above are not very tight. This explains the larger value of the bound gap on termination of the algorithm.

The improving lower and upper bounds are illustrated in Figures 7.2 and 7.3 for the problems NAVY and HUGENAVY respectively. The horizontal axis in both figures is the number of executions of Step 2 of the LQP algorithm. The ability to compute improving upper bounds is an important feature of our approach since computation can be stopped as soon as a reasonable improvement in the upper bound is achieved.

7.4.2 Solving the Naval Assignment Problem

Both problems were solved on the CRAY Y-MP to take advantage of vector capabilities of the CRAY architecture. We give in Table 7.2 the solution statistics of the LQP algorithm. All times are stated in CPU seconds exclusive of input/output. Major iterations refer to

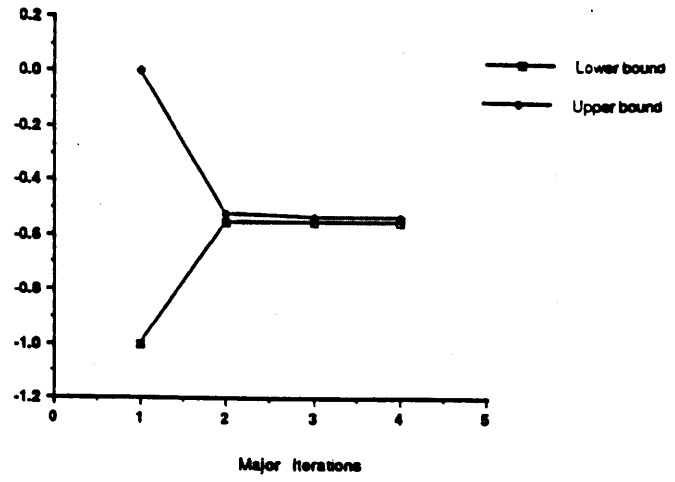


Figure 7.3: Convergence of lower and upper bounds for HUGENAVY

the total number of times Step 1 of the LQP algorithm is executed. It is interesting to note that the larger Navy problem is solved in a time very close to the solution time of the smaller problem. This can be attributed to the larger number of major iterations the algorithm took in the case of the problem NAVY because the smaller problem is more tightly constrained than the larger. Since the iterates generated by the LQP algorithm become only feasible on termination, the previous observation leads to the conclusion that, though much larger in size, HUGENAVY is a relatively easier problem for our method.

Problem	Major iters	Subprob. time	Master time	Total time
NAVY (CRAY Y-MP)	6	45	149	194
NAVY (DEC 5100)	6	132	1428	1560
HUGENAVY(CRAY Y-MP)	2	157	181	276
HUGENAVY(DEC 5100)	NA	NA	NA	NA

Table 7.2: Performance of the LQP algorithm on the Naval personnel assignment problems.

We also experimented with nonlinear versions of NAVY and HUGENAVY problems. We refer to these problems as NAVYQ and HUGENAVYQ where the objective function is a separable quadratic function of the form $\sum_j a_j x_j^2$. For the problem NAVYQ, the coefficients a_j are precisely the coefficients given in the linear model. For the HUGENAVYQ problem the coefficient vector was taken to be identically unity. We report the solution statistics in Table 7.3. Both results were obtained on a CRAY Y-MP.

Problem	Major iters	Subprob. time	Master time	Total time	Lower bound	Upper bound
NAVYQ	18	89	367	456	−273025.94	−252403.
HUGENAVYQ	8	270	867	1137	0.7220×10^8	0.7320×10^8

Table 7.3: Performance of the LQP algorithm on the nonlinear Naval personnel assignment problems.

With both problems, the infeasibility tolerance $\epsilon_{min} = 10^{-5}$ was attained on termination. Using MINOS to solve NAVYQ a feasible solution with objective function value −269515.4 was obtained in 485 CPU seconds on the CRAY Y-MP. This solution is 1% better than the best feasible solution produced by the LQP algorithm in 456 seconds. However, the LQP algorithm produced a more accurate solution on HUGENAVYQ. MINOS was not used for this problem due to the anticipated CPU time and memory requirements.

7.4.3 Comparison and Integration with Linear Programming Solvers

The LQP method delivers quickly an approximate solution to the problem. When higher accuracy is needed a linear programming solver may be used. The smaller problem NAVY is solved with the general purpose linear programming solver MINOS of Murtagh and Saunders [1987]. The statistics are given in Table 7.4. GENOS/LP is outperformed by MINOS on this problem. This is to be expected since the problem is not a large linear program. However, for the larger problem HUGENAVY, MINOS was not able to provide a feasible solution after 1 hour of CPU time on the CRAY Y-MP whereas GENOS/LP solved the problem to an acceptable level of accuracy in less than 5 minutes.

Number of Phase-I pivots	1247
Total number of pivots	2423
CPU time(DEC 5100)	10 mins.

Table 7.4: Performance of MINOS on NAVY.

Interfacing the GENOS/LP system with MINOS may provide MINOS with an advanced starting point. However, since the LQP algorithm is essentially based on an exterior point penalty function, no basis for the problem is readily available. The optimal network basis produced as a result of solving linear network subproblems is input to MINOS. This idea produced a significant reduction in the number of pivots taken by MINOS to reach optimality. A comparison is given below in Table 7.5.

	MINOS	MINOS with advanced start
Number of Phase-I pivots	1247	1750
Total number of pivots	2423	1782

Table 7.5: Performance of MINOS on NAVY using advanced start.

Although a larger number of iterations were needed to produce a feasible solution, the total number of iterations were reduced significantly. Due to the anticipated CPU usage this strategy was not applied to HUGENAVY. We remark that MINOS was not vectorized for the above experiments. We also remark that the general purpose linear programming solver CPLEX solved the problem HUGENAVY in 9 seconds on a CRAY Y-MP, Bixby [1991]. This result outperforms significantly the performance of the LQP algorithm. However, the LQP algorithm is able to handle nonlinear objective functions whereas CPLEX does not have this capability.

7.5 Conclusions

We presented in this chapter a solution method for a large scale application. The LQP method is an exterior point method based on a smooth penalty function and can produce

feasible iterates if an initial feasible point is available. It is able to provide quickly approximate solutions to the problem, which is a valuable feature in a real time planning environment. To achieve higher accuracy, the LQP solution can be used as an advanced start for a general purpose linear programming solver. Although the LQP algorithm is outperformed by CPLEX on the problem HUGENAVY, it is able to handle nonlinear objective functions.

Chapter 8

Conclusions and Extensions

Network optimization enjoyed a great deal of success in the past twenty years. However, this development has not been extended to problems with an embedded network structure and these problems remained a challenge. In this thesis we proposed and developed a solution methodology for the solution of large-scale network-structured optimization problems. We presented numerical experience with (1) large scale multicommodity network flow problems drawn from a military planning application, (2) large network flow problems with side constraints form a Navy personnel assignment application, (3) nonlinear constrained matrix estimation problems and, (4) NETLIB linear programs reformulated as networks with side constraints. The simplicial decomposition/smooth penalty combination we develop in this thesis appears to be an effective and efficient alternative to other solution methods proposed in the literature. The parallel decomposition resulting from the linearization of the subproblem phase and the linear algebra computations led to parallel implementations on both coarse-grain and fine-grain parallel architectures and produced very encouraging results. The solution methodology we developed can handle linear and nonlinear objective functions and its use is not limited by the number and type of side constraint present in the problem. It is also able to handle equality constraints, a distinct superiority with respect to barrier function based methods such as the Schultz and Meyer algorithm and also nonlinear convex constraints.

We want to conclude the thesis by citing a few problems where the smooth penalty approach can be used in the future.

Nonlinearly Constrained Network Problems. As we pointed out, the LQP method is able to handle nonlinear side constraints. There are applications in financial modeling which involve the use of nonlinear constraints. This is an area where our method can be applied.

Convex Nonlinear Programs. The LQP function can be used to solve convex nonlinear programs. To follow this avenue, a generalized truncated Newton method for the solution of the unconstrained problem can be developed. Sun [1990] developed a generalized Newton method for $C1$ functions. An inexact version in the spirit of Dembo et al. [1981] can be developed based on Sun's ideas. We cite this work as prospective future research.

Robust Optimization for the Navy Personnel Assignment Problem. The Navy Personnel Assignment problem involves uncertainty on the supply side which represents the number of trained personnel from different categories. The supply numbers are based on projections and may not be realized, thus creating a shortage of personnel on board ships. This problem can be attacked using the robust optimization methodology developed in Mulvey et al. [1991].

APPENDIX A

We provide a detailed comparison of the lineasearch procedures in Tables A.1 and A.2.

Major iters	Simplicial iters	Function evals	CPU secs
1	4	46	15.37
2	10	295	53.83
3	15	359	94.67
4	19	611	153.18
5	21	792	189.26
6	22	871	205.5
7	23	943	221.16

Table A.1: Solution statistics for PDS1 with quadratic interpolation linesearch.

Major iters	Simplicial iters	Function evals	CPU secs
1	4	94	22.87
2	10	357	79.08
3	15	563	130.38
4	19	879	202.24
5	21	1116	249.00
6	22	1242	273.00
7	23	1345	295.00

Table A.2: Solution statistics for PDS1 with linear–quadratic linesearch.

APPENDIX B

Recall that to form the matrix C we use the following formula

$$C = B^T M B - B^T M \hat{Y} - \hat{Y}^T M B + \hat{Y}^T M \hat{Y}$$

In the implementation of simplicial decomposition, a pointer addressing scheme is used to indicate the active vertices stored in the matrix B. This is necessary to avoid rearranging the columns of the matrix B every time a vertex is added or dropped. The matrix B is stored as a two dimensional array. Each column of B is a vertex which is as long as the number of variables in the problem. The column dimension is determined by the number of vertices at a given iteration. The array LVER is used to point to vertices which are retained. A sparse representation scheme is also used in the computer representation of the second derivative matrix that we denote H. Two arrays IH and JH are used to access the matrix H. We give below the segment of code which computes the matrix C which is stored in upper triangular form by virtue of symmetry. Subroutine HV is used to compute the product of the matrix H with a vector. Subroutine VPROD performs the inner product of two dense vectors.

```
c---- compute the products B'H B and Y' H B
c
c     nv.... number of vertices
c     nvars.. number of variables( = number of variables x number of commodities)
c
c     B is (nvars x nv)
c
      lvnv   = lver (nv)
      do 10 i = 1 , nv - 1
            lvi = lver(i)
c
c---- compute hbi the ith column of the H B product
c
            call hv ( h, ih, jh, b(1,lvi), hbi )
c
```

```
c---- compute xhbi for the ith column of B
c
           call vprod ( b(1,lvnv) , hbi(1) , nvars , xhbi )
c
           do 20 j = 1 , i
               lvj = lver(j)
c
c---- compute the product B'H B for the ith row of B
c
               call vprod ( b(1,lvj) , hbi(1) , nvars , bthbi )
c
c---- add these values to the proper spot in the C matrix
c
               k = j + i*(i-1)/2
               c(k) = bthbi - xhbi
 20        continue
 10  continue
c
c---- compute the product B'H Y
c
      call hv( h , ih, jh, b(1,lvnv), hbi )
c
c.... now compute Y'H Y
c
      call vprod ( b(1,lvnv) , hbi(1) , nvars , xthx )
c
c.... next compute B'H Y for the ith row of B'
c
      do 30 i = 1 , nvm1
            lvi = lver(i)
            call vprod ( b(1,lvi) , hbi(1) , nvars, btihx )
c
```

```
c---- now add these elements to C (same for all hxi)
c
                do 40 j = i , nvm1
                   k        = i + j*(j-1)/2
                   c(k) = c(k) - btihx + xthx
 40             continue
 30  continue
```

Autotasking the program segment above cannot be achieved by simply inserting compiler directives. One way to transform this code into a form suitable for autotasking is given below. This is accomplished by using two subroutines which perform identical operations on different pieces of data. This can be verified in a straightforward manner by going through the code with a simple example. Details such as the dimensioning of the arrays are omitted for the sake of expositional clarity. The autotasking directive **CFPP$ CNCALL** is used to enable concurrent subroutine calls. Notice also the use of Linear Algebra Library routine **SDOT** to compute inner products.

```
      lvnv   = lver (nv)
c
c -- compute B'H B
c
CFPP$ CNCALL
      do 120 i = 1 , nvm1
            lvi = lver(i)
c
c---- compute hbi the ith column of the H B product
c
      call bhb ( h , b(1,lvi) ,  bver, lver, c ,
     &           ih , jh  , i )
 120  continue
c
c
c.... now compute Y'H Y
```

```
c
         call hv (h, ih, jh, b(1,lvnv), hbi)
c
         xthx = sdot ( nvars, bver(1,lvnv), 1, hbi, 1)
c
c.... next compute B' H Y for the ith row of B' for all i
c
CFPP$ CNCALL
      do 130 i = 1 , nvm1
              lvi = lver(i)
       call bhy ( b(1,lvi) , lver, hbi, c, xthx, i)
 130  continue
c
      subroutine bhb ( nv, b, h, v, lver, c,
     &          ih  , jh ,  ivertx)
c
c ----- v is the column of B containing the vertex ivertx
c ----- y is a local array.
c
      lvnv= lver(nv)
c
      do 140 i=1,nvars
 140  y(i) = 0.d0
c
c -- compute y = H v
c
      call hv( h, ih, jh, v, y)
c
       xhbi =  xhbi + sdot(nvars, b(1,lvnv), 1, y, 1)
c
            do 150 j = 1 , ivertx
                lvj = lver(j)
```

```
c
c---- compute the product B'H B for the ith row of B
c
            bthbi =  bthbi + sdot(nvars, b(1,lvj), 1, y, 1)
c
c---- add these values to the proper spot in C
c
                k = j + ivertx*(ivertx-1)/2
                c(k) = bthbi - xhbi
 150         continue
c
      return
      end
c
      subroutine bhy (nv,  v , lver, hbi, c, xthx, ivertx)
c
      nvm1 = nv - 1
c
c -- compute B' H Y for the ivertx-th column of B
c
      btihx = sdot( nvars, v, 1, hbi(1), 1)
c
c---- now add these elements to C (same for all hxi)
c
               do 160 j = ivertx , nvm1
                  k        = ivertx + j*(j-1)/2
                  c(k) = c(k) - btihx + xthx
  160             continue
      return
      end
```

Bibliography

[1] D.P. Ahlfeld, J.M. Mulvey, R.S. Dembo and S.A. Zenios. 1987. Nonlinear programming on generalized networks. *ACM Transactions on Mathematical Software*, Vol 13(4), pp. 350–367.

[2] A. Aho, J.E. Hopcroft and J.D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison–Wesley. Reading Massachusetts.

[3] A.I. Ali, J. Kennington and B. Shetty. 1988. The Equal Flow Problem. *European Journal of Operational Research* 36, 107–115.

[4] A.I. Ali, J.L. Kennington and T.T. Liang. 1988. Assignment with En Route Training of Navy Personnel. Research Report.

[5] S.P. Bradley, A. C. Hax, T. L. Magnanti. 1977. *Applied Mathematical Programming*. Addison-Wesley.

[6] A.A. Assad. 1987. Multicommodity Network Flows: A Survey. *Networks*, 8(1), p.37–92.

[7] I. Ali and J.L. Kennington. 1977. *MNETGN Program Documentation*, Technical Report IEOR 77003, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas.

[8] M. Bazaraa and J. Jarvis. 1977. *Linear Programming and Network Flows*, Wiley and Sons, New York.

[9] D.P. Bertsekas. 1973. Nondifferentiable Optimization via Approximation. *Mathematical Programming Study* 3 (M. Balinski and P. Wolfe, eds.), p. 1–25.

[10] D.P. Bertsekas. 1982. Projected Newton Methods for Optimization Problems with Simple Constraints. *SIAM Journal on Control and Optimization* 20, p. 221–246.

[11] D.P. Bertsekas and E.M. Gafni. 1983. Projected Newton Methods and Optimization of Multicommodity Flows, *IEEE Transactions on Automatic Control* 28(12), p. 1090–1096.

[12] D.P. Bertsekas. 1975. Necessary and Sufficient Conditions for a Penalty Method to be Exact. *Mathematical Programming* 9, p 87–99.

[13] D.P. Bertsekas. 1982 *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, New York, 1982.

[14] D.P. Bertsekas and J.N. Tsitsiklis. 1989. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall.

[15] R.E. Bixby and R. Fourer. 1988. Finding Embedded Network Rows in Linear Programs I. Extraction Heuristics, *Management Science* 34(3), p. 342–376.

[16] R.E. Bixby. 1991. *The Simplex Method: It Keeps Getting Better.* Talk presented at the 14*th* International Symposium on Mathematical Programming, Amsterdam, The Netherlands.

[17] G.G. Brown and R.D. McBride. 1984. Solving Generalized Networks. *Management Science* 30, 1497–1523.

[18] G.G. Brown, G.W. Graves, H. Lange, C. Staniec and R.K. Wood. 1989. Dual Decomposition Methods for Solving Multicommodity Flow Problems, Technical Report, Naval Postgraduate School.

[19] W.J. Carolan, J.E. Hill. J.L. Kennington, S. Niemi and S.J. Wichmann, An Empirical Evaluation of the KORBX Algorithms for Military Airlift Applications. 1990. *Operations Research*, Vol 38(2), p. 240–248.

[20] Y. Censor and A. Lent. 1981. An Iterative Row-action Method for Interval Convex Programming. *Journal of Optimization Theory and Applications* 34, p.321-353.

[21] Y. Censor and S.A. Zenios. 1991. The Proximal Minimization Algorithm with D-functions. *Journal of Optimization Theory and Applications.* (to appear)

[22] M. Chang and M. Engquist. 1986. *GTGEN: A generator for generalized transportation problems*, Research Report CCS 540, Center for Sybernetic Studies, The University of Texas, Austin, (1986).

[23] C. Charalambous. 1978. A Lower Bound for the Controlling Parameter of the Exact Penalty Functions. *Mathematical Programming* 15, p. 278–290.

[24] S. Chen and R. Saigal. 1977. A Primal Algorithm for Solving a Capacitated Network Flow Problem with Additional Linear Constraints. *Networks* 7, 59–79.

[25] C.H.J. Chen and M. Engquist. 1986. A Primal Simplex Approach to Pure Processing Networks. *Management Science* 32, 1582–1598.

[26] Chen R.J. and R.R. Meyer. 1988. Parallel Optimization for Traffic Assignment, *Mathematical Programming* 42, pp. 327–345.

[27] R.H. Clark and R.R. Meyer. 1987. *Multiprocessor algorithms for generalized network flows.* Computer Sciences Technical Report # 739, University of Wisconsin–Madison.

[28] CRAY X–MP Multitasking Programmer's Reference Manual. 1987. CRAY Research Inc.

[29] CRAY Autotasking Training Workbook. 1989. CRAY Research Inc.

[30] G.B. Dantzig. 1963. *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey.

[31] R.S. Dembo, S.C. Eisenstat and T. Steihaug. 1981. Inexact Newton Methods. *SIAM Journal of Numerical Analysis* 19, p. 400-408.

[32] R.S. Dembo and T. Steihaug. 1983. Truncated Newton algorithms for large–scale unconstrained optimization. *Mathematical Programming* 26, p. 190–212.

[33] R.S. Dembo. 1987. A Primal Truncated Newton Algorithm with Application to Non-linear Network Optimization. *Mathematical Programming Study* 31, p. 43–72,

[34] R.S. Dembo and R. Anderson. 1989. *An Efficient Linesearch for Convex Piecewise-Linear/Quadratic Functions*, Algorithmics Inc. Working Paper 89.02, Toronto, Canada.

[35] R.S. Dembo and J. Klincewicz. 1985. Dealing with Degeneracy in Reduced Gradient Algorithms, *Mathematical Programming* 31, p. 357–363.

[36] R.S. Dembo, J.M. Mulvey and S.A. Zenios. 1989. Large Scale Nonlinear Network Models and Their Application, *Operations Research* 37(3), p. 353–372.

[37] R.S. Dembo and U. Tulowitzki. 1988. Computing Equilibria on Large Multicommodity Networks, An Application of Truncated Quadratic Programming Algorithms, *Networks*, 18, p. 273–284.

[38] R.S. Dembo. 1988. *Towards the Solution of Large Scale Nonlinear Programs*, Working Paper, University of Toronto, Toronto, Canada.

[39] J.E. Dennis,Jr. and R.B. Schnabel. 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice–Hall. New Jersey.

[40] L.F. Escudero. 1986a. A motivation for using the truncated Newton approach in a very large scale network problem. *Mathematical Programming Study* 26, p.240–244.

[41] L.F. Escudero. 1986b. Performance evaluation of independent superbasic sets on nonlinear replicated networks. *European Journal of Operations Research* 23, p. 343–355.

[42] B. Feinberg. 1989. Coercion Function and Decentralized Linear Programming. *Mathematics of Operations Research* 14 (1), p. 177-187.

[43] A.V. Fiacco and G.P. McCormick. 1968. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley and Sons, New York.

[44] A.B. Gamble, A.R. Conn and W.R. Pulleyblank. 1991. A Network Penalty Problem. *Mathematical Programming* 50, 53–74.

[45] A. Geoffrion. 1977. Objective Function Approximations in Mathematical Programming. *Mathematical Programming* 13. p. 23–37.

[46] P. Gill, W. Murray, and M.H. Wright. 1981. *Practical Optimization.* Academic Press. London and New York, 1981.

[47] F. Glover, J. Hultz, D. Klingman and J. Stutz. 1978. Generalized Networks: A Fundamental Computer Based Planning Tool. *Management Science* 24, 1209–1220.

[48] F. Glover and D. Klingman. 1981. The Simplex SON Algorithm for LP/Embedded Network Problems. *Mathematical Programming Study* 15, 148–176.

[49] F. Glover, D. Klingman and N. Phillips. 1990. Netform Modeling and Applications. *Interfaces.* 20:4, 7–27.

[50] M.D. Grigoriadis and W.W. White. 1972. A Partitioning Algorithm for the Multicommodity Network Flow Problem, *Mathematical Programming* 3, p. 157–177.

[51] D.W. Hearn, S. Lawphongpanich and J.A. Ventura, Restricted Simplicial Decomposition: Computation and Extensions. 1987. *Mathematical Programming Study* 31, p. 99–118.

[52] W.D. Hillis. 1985. *The Connection Machine.* The MIT Press. Cambridge. Massachusetts.

[53] J.K. Ho and S.K. Gnanendran. 1989. *Distributed Decomposition of Block-Angular Programs on a Hypercube Computer*, Working Paper, Management Science Department, University of Tennessee.

[54] B. Von Hohenbalken. 1977. Simplicial Decomposition in Nonlinear Programming Algorithms, *Mathematical Programming* 13, p. 49–68.

[55] C.A. Holloway. 1974. An Extension of the Frank–Wolfe Method of Feasible Directions, *Mathematical Programming* 6(1), p. 14–27.

[56] N. Karmarkar. 1984. A New Polynomial Time Algorithm for Linear Programming, *Combinatorica* 4, p. 373–395.

[57] J.L. Kennington. 1978. A Survey of Linear Multicommodity Network Flows, *Operations Research* 26(2), p. 209–236.

[58] J.L. Kennington and R.V. Helgason. 1980. *Algorithms for Network Programming.* Wiley–Interscience. New York.

[59] J.L. Kennington and M. Shalaby. 1977. An Effective Subgradient Procedure for Minimal Cost Multicommodity Flow Problems, *Management Science*, 23(9), p. 994–1104.

[60] D. Klingman, M. Mead and N.V. Phillips. 1984. Network Optimization Models for Military Manpower Planning. *Operational Research'84.* J.P. Brans (Editor).

[61] J. Koene. 1982. Minimal Cost Flow in Processing Networks, A Primal Approach. Ph.D. Thesis. Eindhoven University of Technology. Eindhoven. The Netherlands.

[62] I.A. Krass and T.J. Thompson. 1990. Mathematical Formulation of EDPROJ – Readiness Connection, Navy Research Report.

[63] I.A. Krass, T.T. Liang and T.J. Thompson. 1988. Quantifying the Impact of Moving Budget on Navy Enlisted Personnel Unit Readiness, Navy Research Report.

[64] L.J. LeBlanc, E.K. Morlok and W.P. Pierskalla. 1975. An Efficient Approach to Solving the Road Network Equilibrium Traffic Assignment Problem, *Transportation Research*, 9, p. 309–318.

[65] Z. Liu. 1988. *Nonlinear Pricing and Decomposition Techniques with Application to Multicommodity Network Flows*, Thesis No.155, Department of Mathematics, Linköping Institute of Technology, Linköping, Sweden.

[66] D.G. Luenberger. 1970. Control Problems with Kinks. *IEEE Transactions on Automatic Control* 15, p.570-574.

[67] D.G. Luenberger. 1984. *Linear and Nonlinear Programming*, Addison–Wesley, Reading, MA.

[68] I. Lustig, R. Marsten, D. Shanno. 1992. Computational Experience with a Primal-Dual Interior Point Method. *Linear Algebra and Its Applications*, (to appear).

[69] R. Marsten, I. Lustig, R. Subramanian, M. Saltzman, D. Shanno. 1990. Interior Point Methods for Linear Programming, *Interfaces* 20(4), p. 105–116.

[70] R.D. McBride. 1985. Solving Embedded Generalized Network Problems. *European Journal of Operational Research* 21, 82–92.

[71] R.R. Meyer and S.A. Zenios. 1988. *Parallel Optimization on Novel Computer Architectures*, Annals of Operations Research, Vol. 14, J.C. Baltzer AG, Scientific Publishing Company, Basel, Switzerland.

[72] J.M. Mulvey. 1978. Testing of a Large Scale Network Optimization Program. *Mathematical Programming* 15, 291–315.

[73] J.M. Mulvey and H. Vladimirou. 1991. *Solving Multistage Stochastic Networks: An Application of Scenario Aggregation. Networks*, (to appear).

[74] J.M. Mulvey and S.A. Zenios. 1985. Solving Large Scale Generalized Networks. *Journal of Information and Optimization Sciences* 6, 95–112.

[75] J.M. Mulvey, S.A. Zenios and D.P. Ahlfeld. 1990. Simplicial Decomposition for Convex Generalized Networks, *Journal of Information and Optimization Sciences* , 11(2), pp. 359–387.

[76] J.M. Mulvey, R.J. Vanderbei and S.A. Zenios. 1991. Robust Optimization of Large Scale Systems: General Modeling Framework and Computations. Decision Sciences Department Report 91-06-04. University of Pennsylvania. Philadelphia. PA 19104.

[77] B. Murtagh and M.A. Saunders. 1987. MINOS 5.1 User's Guide. Report SOL 83-20R, December 1983, revised January 1987, Stanford University.

[78] B. Murtagh and M. Saunders. 1978. Large–scale linearly constrained optimization. *Mathematical Programming* 14, pp. 41–72.

[79] S.G. Nash and A. Sofer. 1989. Block Truncated Newton methods for parallel optimization. *Mathematical Programming* 45, pp. 529–546.

[80] S.N. Nielsen and S.A. Zenios. 1990. *A Massively Parallel Algorithm for Nonlinear Stochastic Network Programs*, Working Paper, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, PA. 19104.

[81] S.N. Nielsen and S.A. Zenios. 1991a. Massively Parallel Algorithms for Singly Constrained Nonlinear Programs. *ORSA Journal on Computing* 4 (to appear).

[82] S.N. Nielsen and S.A. Zenios. 1991b. *Proximal Minimizations with D-Functions and the Massively Parallel Solution of Linear Network Programs.* Department of Decision Sciences. Report 91–06–05. University of Pennsylvania. Philadelphia. PA 19104.

[83] M.Ç. Pınar and S.A. Zenios. 1992. *Parallel Decomposition of Multicommodity Network Flows using Smooth Penalty Functions. ORSA Journal on Computing* 4 (to appear).

[84] L. Qi and J. Sun. 1990. A Nonsmooth Version of Newton's Method and an Interior Point Algorithm for Convex Programming. Research report.

[85] R. T. Rockafellar. 1970. *Convex Analysis.* Princeton University Press.

[86] R.T. Rockafellar. 1976a. Augmented Lagrangians and Applications to Proximal Point Algorithms in Convex Programming. *Mathematics of Operations Research* 1, p.97-116.

[87] R.T. Rockafellar. 1976b. Monotone Operators and the Proximal Point Algorithm. *SIAM Journal on Control and Optimization* 14, p.877-898.

[88] R.T. Rockafellar. 1984. *Network Flows and Monotropic Optimization.* Wiley–Interscience. New York.

[89] R.E. Rosenthal. 1981. A nonlinear network flow algorithm for maximizing the benefits in a hydroelectric power system. *Operations Research* 29, p. 763–786.

[90] G.L. Schultz and R.R. Meyer. 1991. *An Interior Point Method for Block Angular Optimization, SIAM Journal on Optimization* 1, p.583-602.

[91] R. Schneur. 1991. Scaling Algorithms for Multicommodity Flow Problems and Network Flow Problems with Side Constraints. Ph.D Thesis. Department of Civil Engineering. Massachussetts Institute of Technology.

[92] R. Setiono. 1989. *Dual Proximal Interior Point Methods for Linear Programming,* Technical Report, Department of Computer Science, University of Wisconsin–Madison.

[93] R.E. Stone. 1988. *An Algorithm for Solving Network Programs with Side Variables*, Working Paper, A T & T Bell Laboratories, Holmdel, NJ 07733.

[94] R.E. Tarjan. 1972. Depth–first search and linear graph algorithms. *SIAM Journal on Computing* 1, pp. 146–160.

[95] Ph.L. Toint and D. Tuyttens. 1990. On Large Scale Nonlinear Network Optimization, *Mathematical Programming Series B* 48, p.125–159.

[96] J.A. Tomlin, Minimum Cost Multicommodity Network Flows. 1966. *Operations Research* 14, p. 45–51.

[97] K. Truemper. 1975. Note on Finite Convergence of Exterior Penalty Functions. *Management Science* 21, p. 600–606.

[98] R.D. Wollmer. 1972. Multicommodity Networks with Resource Constraints: The Generalized MUlticommodity Network Flow Problem, *Networks* 1, p. 245–263.

[99] Y. Wu and T.G. Lewis. 1989. *Parallel Algorithms for Decomposed Linear Programs*, Working Paper, Computer Science Department, Oregon State University.

[100] I. Zang. 1981. Discontinuous Optimization By Smoothing, *Mathematics of Operations Research* 6 (1), p. 140–152.

[101] I. Zang. 1980. A Smoothing-out Technique for Min-Max Optimization. *Mathematical Programming* 19, p. 61–77.

[102] S.A. Zenios, M.Ç. Pınar. 1990. *Solving Large Scale Multicommodity Networks using Linear–Quadratic Penalty Functions*, Extended Abstract, to appear in *Proceedings of the NATO Advanced Study Institute in Combinatorial Optimization*, Springer–Verlag.

[103] S.A. Zenios and Y. Censor. 1991. Massively Parallel Row-Action Algorithms for Some Nonlinear Transportation Problems. *SIAM Journal on Optimization* 1, p.373-400.

[104] S.A. Zenios and R.A. Lasken. 1988. Nonlinear Network Optimization on a Massively Parallel Connection Machine. *Annals of Operations Research* 14, p.147-165.

[105] S.A. Zenios and J.M. Mulvey. 1987. *User's Guide to GENOS 1.0: A Generalized Network Optimization System.* Department of Decision Sciences Report 87–12–03, University of Pennsylvania (1987).

[106] S.A. Zenios and J.M. Mulvey. 1988. Vectorization and Multitasking of Nonlinear Network Programming Algorithms, *Mathematical Programming* 42, p.449–470.

[107] S.A. Zenios and J.M. Mulvey. 1986. Nonlinear Network Programming on Vector Supercomputers: A Study on the CRAY X–MP, *Operations Research*, 34, p.667–682.

[108] S.A. Zenios. 1989. Parallel Numerical Optimization: Current Status and An Annotated Bibliography, *ORSA Journal on Computing* 1, p.20–43.

[109] S.A. Zenios, A. Drud and J.M. Mulvey. 1989. Balancing Large Social Accounting Matrices with Nonlinear Network Programming. *Networks* 19, 569–585.

[110] S.A. Zenios and M.Ç. Pınar. 1992. Parallel Block Partitioning of Truncated Newton for Nonlinear Network Optimization, *SIAM Journal on Scientific and Statistical Computing*, (to appear).

[111] S.A. Zenios. 1991. On the Fine-Grain Decomposition of Multicommodity Transportation Problems, *SIAM Journal on Optimization* 1, p.643-669.

[112] S.A. Zenios, R.S. Dembo and M.Ç. Pınar. 1990. *A Smooth Penalty Function Algorithm for Network Structured Problems*, Department of Decision Sciences Report 90-12-05, University of Pennsylvania, Philadelphia, PA. 19104.

[113] S.A. Zenios, R. Qi and P.K. Armstrong. 1991. Coercion Methods for Decomposition of Nonlinear Programs. *Networks*, (to appear).